Trabajo de Fin de Grado
Ingeniería Aeroespacial

Sistema de posicionamiento en interiores basado en visión artificial de aplicación aeronáutica

Autor: Miguel Ángel Alcaide Márquez

Tutor: María Ángeles Martín Prats

Dpto. de Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado
Ingeniería Aeroespacial

Sistema de posicionamiento en interiores basado en visión artificial de aplicación aeronáutica

Autor:
Miguel Ángel Alcaide Márquez

Tutores:
María Ángeles Martín Prats
Profesor Titular

Dpto. de Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado: Sistema de posicionamiento en interiores basado en visión artificial de aplicación aeronáutica

Autor: Miguel Ángel Alcaide Márquez
Tutor: María Ángeles Martín Prats

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Tras estos 4 años aquí en Sevilla he aprendido mucho no solo lo que significa ser un ingeniero ni mucho menos. Si no lo que significa el esfuerzo, la motivación de superarse cada día, ser capaz de ayudar y dejarse ayudar por los que te rodean pues tienen grandes cosas que ofrecer.

Con este TFG culmino mis 4 años de carrera en Sevilla solo me quedan palabras de agradecimiento para los que me han rodeado. Mi familia ha sido un pilar fundamental pues cuando creía que todo esto me superaba, cuando todo pensaba que iba a salir mal, allí estaban y gracias por soportarme cuando os hablaba de cosas que no entendíais por eso gracias abuelete, mama, tito Manolo y Victoria os quiero mucho.

Pero sobre todo quiero agradecer a mis amigos aquí en Sevilla en especial a Javi y Mendiola. Que decir de vosotros fuisteis mis primeros amigos en Sevilla. Aquellos interminables días de estudio haciendo problemas en la pizarra, mis encabezamientos con Mendiola por errores mios y esas tertulias infinitas en nuestros cuartos a cada cual mas bizarra. Mucho vivido, reído, sufrido, disfrutado pero siempre juntos, os agradezco poder haberos acompañado estos años y los que quedan porque sin vostros no habria logrado lo que he logrado tanto como persona como en mis estudios.

También quiero agradecer a mi gran amigo Quique, cuantos años hacen ya desde que te conocí, me acuerdo perfectamente un dia de los primeros del curso en el final de aquella clase, doy gracias de coincidir con tan grande persona pues de estos años saco tener una amistad como ninguna otra. Esos días de playa jugando al voley los paseos infinitos en los que dábamos mas de una vuelta a Cádiz entero, esas una mas hasta la madrugada esas conversaciones rocambolescas y cuanto menos fidedignas. Gracias por acompañarme aun estando lejos estos 4 años en Sevilla y gracias por convertirte en lo que considero mas que un amigo, un hermano pues tu has estado siempre junto a mi en lo dificil y facil de esta travesía y que se que seguirá siendo así, gracias.

Tampoco me puedo olvidar de mis amigos del colegio que tanto quiero Cristian, Chechu, Razquin, Agustin, Linde, Drolu, Antonio y mis amigos de la universidad Dani, Ale, Javieta, Victor, Jacobo son tantas personas y tanto que agradecer que no puedo decirlo todo, por lo que gracias.

Miguel Ángel Alcaide Márquez
Sevilla, 2021

Resumen

La algoritmia relacionada con sistemas de posicionamiento indoor es muy útil en una gran cantidad de campos que abarcan desde la industria 4.0 hasta la industria creativa. Aunque cabe remarcar que lo es mas en temas de logística y ahorro de costes, tanto en tiempo como en mano de obra. Con el asentamiento del uso de drones tanto en la industria como en un ámbito mas doméstico, han aparecido muchos estudios acerca de sus posibles aplicaciones.

Este proyecto es el resultado de los muchos estudios acerca de posibles aplicaciones. Aparece un serio problema cuando se quieren usar en interiores, este problema es ubicar el drone con la suficiente precisión. Por ello este trabajo tiene como objetivo crear un sistema de posicionamiento indoor de aplicación aeronáutica.

Se analizan las diferentes tecnologías existentes y se escoge la realidad aumentada. A partir de este punto se eligen las herramientas necesarias para el desarrollo de este sistema.El cual necesitara del propio algoritmo de mapeado, así como una interfaz para posicionar el drone.

La algoritmia de mapeado se basa en crear una nube de puntos a través de la reconstrucción de la ubicación de los marcadores de realidad aumentada, usando las distancias relativas entre ellos. Y en cuanto a la interfaz gráfica, es una aplicación sencilla con los instrumentos mas básicos que debería disponer un piloto para ser capaz de monitorizar la trayectoria del drone.

Abstract

Computer science of indoor positioning systems not only is truly useful on many study fields such as Industry 4.0 and Creative Industries, but also on logistics, cost saving and labor force.

The rich literature about possible indoor positioning systems applications helped the author as to write this paper. This literature concludes that positioning the drone accurately enough is a serious issue. Therefore, this paper aims to create an accurate aeronautic indoor positioning system. The author analyzed state-of-the-art technologies and decided to base this paper on augmented reality. The author built a mapping algorithm, and a graphic interface in order to position the drone as to the system development.

The mapping computer science is based on a point cloud creation through the augment reality maker location using the relative distances between them. The graphic interface is an intuitive application which includes the most basic tools a drone pilot should have as to monitor the drone track.

Índice

1 Introducción

1.1 Objeto

Realizar un programa capaz de realizar un mapeo de instalaciones indoor, como puede ser una nave de la FAL, con el objetivo de obtener un posicionamiento indoor basado en dicho mapeo, solucionando los problemas subyacentes a otro tipo de posicionamiento solo viables en exteriores como el GPS diferencial u otros que requieren un gran coste de instalación como los sistemas UWB explicados a lo largo del documento.

1.2 Motivaciones

El avance tecnológico en las industrias cada vez esta mas presente, de ahí que aparezca el concepto de la industria 4.0.

Dicho concepto queda bien definido con el siguiente texto extraido de Deloitte España que dice: *"Industry 4.0 signifies the promise of a new Industrial Revolution—one that marries advanced production and operations techniques with smart digital technologies to create a digital enterprise that would not only be interconnected and autonomous but could communicate, analyze, and use data to drive further intelligent action back in the physical world."* [2] Lo que en resumidas cuentas significa la implementación de todos estos avances a la industria.

Con el auge de los drones en la industria para la inspección de infraestructuras como aerogeneradores, lineas de alta tensión e instalaciones petrolíferas. Se mejoraron muchos aspectos como la eficiencia y la seguridad laboral debido a que operaciones costosas y peligrosas como podría ser revisar las palas de un aerogenerador se solucionan con un operario de drone y otro que manejara la cámara y realizara la inspección, ahorrando el factor de riesgo de la operación y acelerando los procesos.

Todas estas operaciones están basadas en vuelos en exteriores con amplio espacio de maniobra y otros sistemas de localización como puede ser el GPS diferencial, donde por norma general no necesitas precisión de centimétros. En el caso de interiores existe el problema de que localizar un UAV es bastante mas complicado sin tener en cuenta que el espacio es menor y por lo general la precisión requerida será mayor. Debido a esto, el uso de sistemas de UAVs en interiores no esta tan desarrollado. Aunque eso no quiere decir que no sea igual de útil o mas, debido a este problema actualmente se están lanzando proyectos a nivel europeo para el desarrollo de sistemas de posicionamiento indoor, un claro ejemplo es el proyecto AiRT [1].
El proyecto AiRT tiene como objetivo principal crear el primer sistema RPAS indoor diseñado específicamente para uso profesional.

Actualmente existen proyectos para la implementación del uso de drones en la industria 4.0 como puede ser en la planta de Airbus Defence and Space en Puerto Real con el objetivo de transportar piezas y herremaientas por la nave así como evitar el FOD (Foreing Object Damage) encontrando las herramientas perdidas por la nave al final de la jornada de trabajo para evitar su olvido dentro de componentes de la aeronave.Tambien existen otros como el de PROSEGUR la empresa dedicada a la seguridad que se basaba en usar sistemas RPAS para la vigilancia de las naves [3].

Por todo ello para poder hacer uso de estos aparatos en el interior de las fabricas es necesario un sistema de posicionamiento suficientemente preciso para evitar colisiones y un manejo preciso de las trayectorias con el objetivo de hacer estos sistemas totalmente autónomos. Si este problema fuera solucionado por completo daría como resultado, el uso de drones en interiores como solución viable de muchas aplicaciones, que hasta ahora se hacia de forma manual o con otros sistemas mas costos o lentos.

Abrir la puerta al uso de drones en interiores daria paso a un gran sector dentro de la industria aeronáutica, ya existe un amplio campo de aplicación para exteriores y con esto se lograría aumentar mas.

Entrando en detalle con el sistema aquí planteado el uso de realidad aumentada como alternativa de posicionamiento tiene muchas ventajas frente a las tecnologías existentes, aunque para ello habría que obtener el máximo de esta aplicación.

Por lo que observando ventajas e inconvenientes de esta tecnología:

Ventajas:

- **Bajo coste:** Debido a que el sistema se basa en realidad aumentada solo es necesario el uso de una cámara suficiente precisa y referencias visuales, que al fin y al cabo son pegatinas con un patrón impreso. Por lo que el coste es ínfimo en comparación con otras tecnología como la basada en UWB explicado en el capitulo 2.1
- **Bajo nivel de computación:** Al ser un sistema muy sencillo que no aplica algoritmos complejos como puede ser los usados para vSLAM, la necesidad de incorporar un procesador de alta potencia no es necesario puesto que esto implica un aumento de peso y consumo considerable. Pudiendo usar el resto de la capacidad de calculo para otros menesteres.
- **Alta precisión:** Los sistemas de posicionamiento basados en visión artificial son una disciplina en desarrollo pero que actualmente da muy buenos resultados en comparación con el resto de sistema a excepción de usar sistemas lidar 3D que aunque son muy precisos son extremadamente caros y pesados, por lo que teniendo en cuenta esto, esta opción no es la mas viable para interiores
- **Rápida instalación:** Solo necesita de ubicar espaciadamente en las instalaciones las referencias visuales, realizar un vídeo de estas y ejecutar el programa de mapeo. Mientras que otros sistemas requieren de instalar diferentes cámaras por la nave o emisores y receptores de señales electromagnéticas o ultrasonidos.

Inconvenientes:

- **Visión directa:** Es necesario que el drone este en todo momento en linea directa con al menos una referencia visual y que esta no se encuentre extremadamente lejos puesto la precisión va sufrir en función de este parámetro
- **Dependencia de las condiciones lumínicas:** Dependiendo de las condiciones lumínicas la detección sera mas precisa o todo lo contrario, por ejemplo una nave con escasa iluminación puede que ciertos marcadores no puedan ser detectados ocasionando una perdida de precisión o en el peor de los casos ser incapaz de posicionarse.
- **Dependencia con la toma dinámica de imágenes:** Esto quiere decir que si el drone se mueve a una gran velocidad o la cámara esta sometida a muchas vibraciones o movimientos bruscos las imágenes que se toman por ende son peores lo que va en detrimento de la detección de los marcadores

Como conclusión de este análisis rápido vemos las claras ventajas de este sistema frente a unos inconvenientes técnicos que podrían ser solucionados con sistemas de soporte aunque el mayor de estos problemas es la dependencia de las condiciones lumínicas

1.3 Estructura del documento

- **Introducción:** Capitulo anterior en el que se muestran los motivos y objetivos de este trabajo.

- **Estado del arte:** Orientado a dar conocer la tecnologías existentes entre ellas UWB, WiFi, Bluetooth, Utrasonidos, RFID y visión artificial. Conocidos las tecnologías sus ventajas y metodos de funcionamiento se escoge la mas adecuada al proyecto.
- **Visión artificial:** En este capitulo se describe en que se basa la visión artificial así como los elementos necesarios para poder implementarla junto con las posibles aplicaciones.
- **Algoritmia de mapeado:** Se describe en que consiste el programa realizado, se explica cada parte del programa en secciones diferentes, siendo estas, recopilación de datos, posiciones relativas, grafo y distancias globales.
- **Interfaz gráfica:** Se explican los diferentes elementos que esta interfaz posee, así como estos se han llegado a implementar, hablando de los conceptos de memoria compartida, multiproceso y el uso de la librería que ha permitido crear la interfaz.
- **Resultados numéricos:** Se muestran los resultados para diferentes números de marcadores y disposiciones, con el objetivo de sacar conclusiones con respecto a su precisión. También se analizan los errores de base que trae detectar los marcadores.
- **Conclusiones y lineas futuras:** Una vez realizadas todas las comprobaciones y obtenidos los resultados del desempeño de los programas, se realizan las conclusiones sobre este. Y por ultimo se enumeran propuestas de mejora y posibles investigaciones futuras, fruto de lo ya realizado.

2 Estado del arte

En la actualidad existen diferentes soluciones para el posicionamiento indoor, estos sistemas llamados IPS (Indoor Position Systems) usan diferentes tecnologías todas en desarrollo, aunque algunas ya están consolidadas a nivel industrial, de las cuales hay que destacar las siguientes.

- WiFi
- Bluetooth
- UWB (Ultra Wide Band)
- Ultrasonidos
- RFID (Radio Frequency Identification)
- Visión artificial

Existen muchas metodologías y por ello es necesario profundizar en cada una de ellas para discernir cual es la mas apropiada para cumplir nuestros objetivos.

2.1 UWB/WiFi/Bluetooth

Para el posicionamiento basado en tecnologías en UWB, WiFi y Bluetooth se necesitará una buena cobertura en toda la instalación, siendo por tanto obligatorio tener los routers suficientes para abarcar todo el espacio objeto de estudio.

La única diferencia principal entre WiFi y Bluetooth es alcance y precisión, todo esto se debe a la frecuencia en la que funcionan, por lo general mediante WiFi tienes mas alcance pero menor precisión que con tecnología Bluetooth pero los métodos de localización son iguales.

El método mas preciso y con mejor alcance es UWB que usa ondas de alta frecuencia para una mayor precisión y alcance.

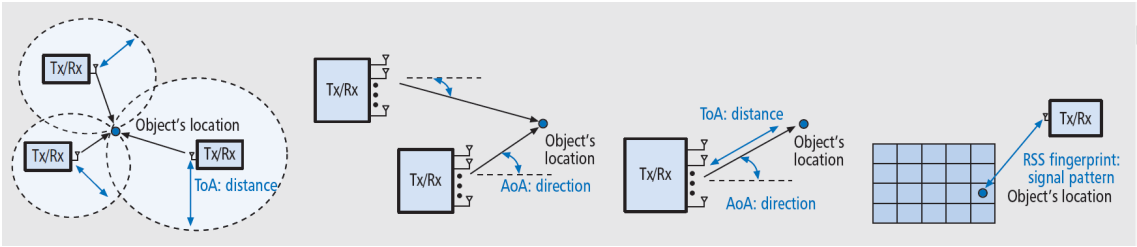

Figura 2.1 Métodos de localización basado en WiFi y Bluetooth.

Estas tecnologías usan 4 métodos principalmente, algunos de ellos se repetirán más tarde en los diferentes métodos.

2.1.1 ToA (Time of arrival approach)

Esto se basa como su nombre indica en el tiempo que tarda una señal en llegar del emisor al receptor, conociendo este tiempo y multiplicando por la velocidad de la luz. Por lo general esta metodología necesita de varios routers con el objetivo de realizar una trilateración de la localización del receptor.

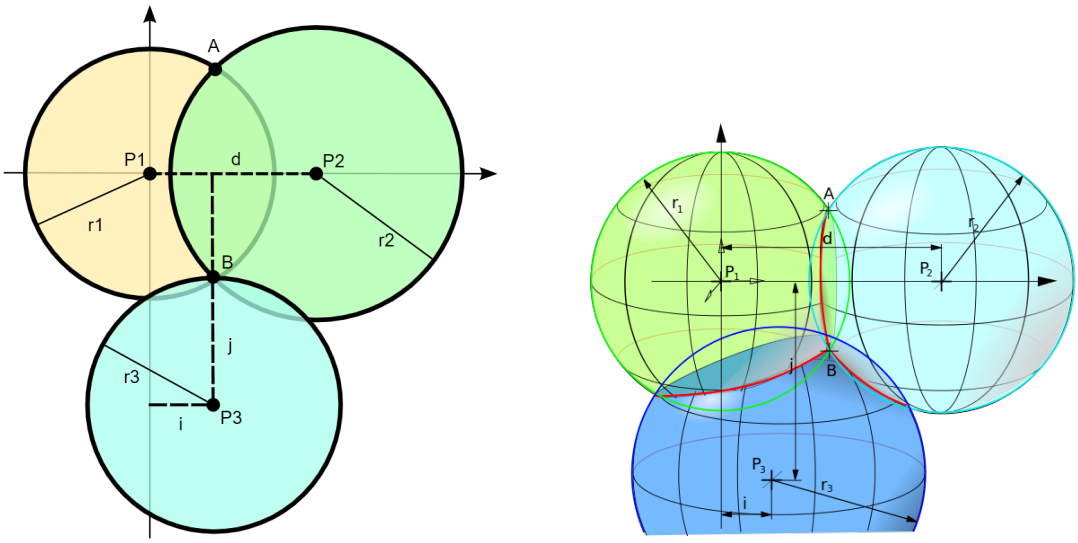

Figura 2.2 Trilateración 3D.

Se trata de realizar una trilateración 3D obteniendo la intersección de 3 esferas para calcular la posición, resolviendo el sistema de 3 ecuaciones aquí expresados.

$$r_1 = x_2 + y^2 + z^2$$

$$r_2 = (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2$$

$$r_3 = (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2$$

Siendo los elementos $(x_i y_i z_i)$ las respectivas posiciones de los centros de la segunda y tercera esfera respectivamente.

2.1.2 AoA (Angle of arrival approach)

En este caso se basa en la triangulación por la cual conocida la dirección de la que proviene la señal podemos llegar a establecer la posición del receptor.

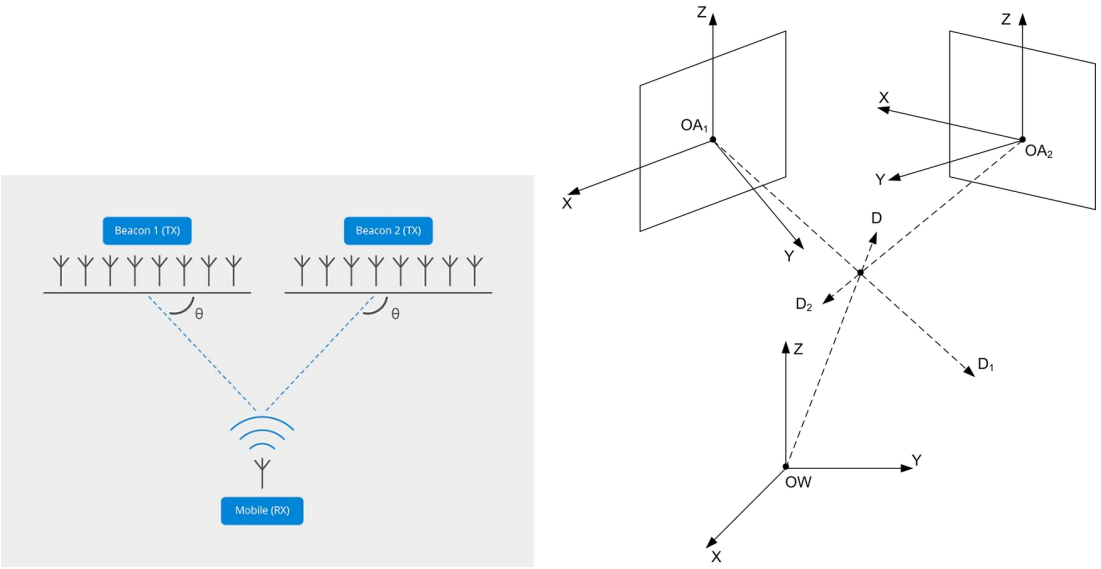

Figura 2.3 Triangulación 3D.

Al igual que antes existe un caso tridimensional y bidimensional solo queda resolver la intersección de estas direcciones para calcular la posición.

2.1.3 hybrid ToA/AoA approach

Usando ambas tecnologías nos permite localizar el receptor unica y exclusivamente con un emisor pues conocida la distancia a este y el ángulo que forma con el emisor queda definida la posición.

2.1.4 RSS fingerprint

Gracias al factor RSS (Received Signal Strength) que es una medida de intensidad de señal nos permite realizar un mallado de los diferentes valores de RSS que se miden en las instalaciones.

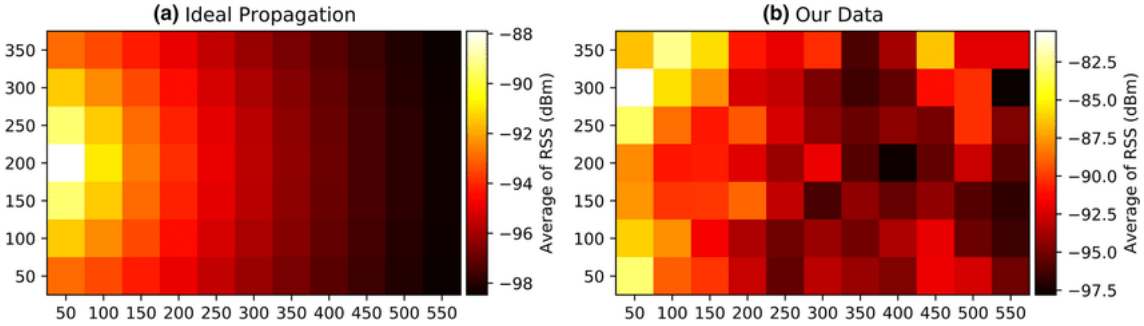

Figura 2.4 Modelo teórico y experimental de propagación.

Como se puede apreciar un modelo real difiere del teórico. El modelo teórico determina una caída exponencial con la distancia de la intensidad de señal, pero esto se ve perturbado debido a las interferencias en el medio, como pueden ser paredes, otras ondas, elementos metálicos y otro tipo de interferencias, ya que el modelo teórico es diferente se realiza un fingerprint que es un mallado de estas RSS. Con cada instalación de un router se genera un mallado, y teniendo en cuenta esto, cuantos mas router tengamos, podremos obtener una solución mas exacta, debido que pocas localizaciones cumplirán los requisitos de todos los routers.

2.2 Ultrasonidos y RFID

Ambos sistemas son mas simples que los anteriores y por tanto poseen una menor complejidad en cuanto a computación se refiere pues se basan en ToA y en RSS fingerprint respectivamente aunque a menudo para

los RFID solo se usa el modelo teórico de propagación de señal y no fingerprints ya que normalmente la targeta RFID es movil.

2.2.1 Ultrasonidos

Existen varios emisores enviando cadenas de pulsos de ultrasonidos con los cuales puede conocer el tiempo que tardan en llegar al receptor, creando así de nuevo un algoritmo de trilateración, pero para ello requiere visión directa emisor receptor y espacios muy amplios sin objetos que hagan que dichos pulsos reboten, esta tecnología es muy precisa dando medidas de 5 cm de error en un rango de 20 m es el sistema mas preciso hasta hora descrito, pero con claras desventajas.

2.2.2 RFID

Existen de varios tipos pero las de interés para este proyecto son las pasivas puesto que no requieren de revisión continua, las tarjetas RFID poseen una antena y un CI (circuito integrado) el cual se alimenta cuando recibe una señal a una frecuencia determinada siendo capaz de transmitir información, dando de nuevo lugar al concepto de RSS pero para tarjetas RFID por lo que usando o bien trilateración o fingerprints se localiza el emisor de estas señales.

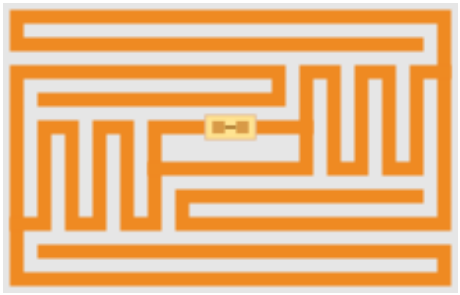

Figura 2.5 Tarjeta RFID.

2.3 Visión artificial

Este es un campo en auge que abarca desde los sistemas mas sencillos a los mas complejos pero el mas consolidado es el visual SLAM

Visual SLAM es un sistema que crea un mapa y se ubica en el de forma paralela. Es algo de una gran complejidad que requiere de una capacidad de procesamiento alta y algoritmos del mas alto nivel.

Se basan en obtener sucesivas imágenes 3D, creando mapas con la forma de las instalaciones y una vez realizados, mediante algoritmos basados en reconocimiento de patrones e inteligencia artificial, localizan el punto mas se adecue al mapa generado.

Para generar este mapa existen diferentes formas:

- Visión estereoscópica
- Camaras RGB-d

El primer método es el mas intuitivo, pues es como cualquier ser vivo percibe la profundidad de lo que vemos, esto se debe a la diferente percepción de una misma imagen por dos cámaras separadas.

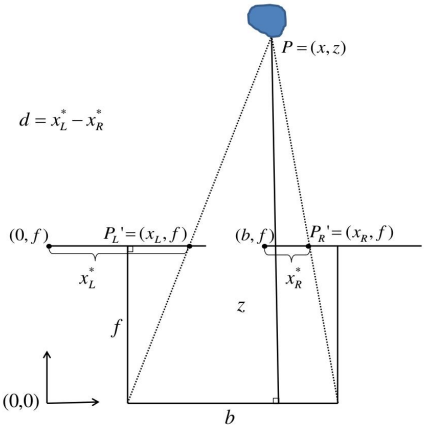

Figura 2.6 Calculo de posición visión estereoscópica.

Las cámaras RGB-d funcionan mediante una imagen RGB y otra que mide la profundidad de cada píxel pero son bastante imprecisas por lo que se suele usar el primer método aunque requiere de mas coste computacional, ya que hay que localizar cada píxel de la primera imagen con cada píxel de la segunda, interviniendo algoritmos de búsqueda complejos o algoritmos basados en redes neuronales.

Observando como funciona el sistema vSLAM y teniendo en cuenta la existencia de tecnologías basadas en AR(Augmented Reality) que permiten conocer la distancia de la cámara al marcador de AR se podría generar un mapa si en las instalaciones existen estratégicamente distribuidos estos marcadores.

Recapitulando todos los métodos descritos anteriormente para la localización necesitan grandes infraestructuras para su correcto funcionamiento a excepción de los dos últimos, el RFID y la visión artificial.Por lo que teniendo en cuenta que la visión artificial tiene un mayor campo de investigación y avance además de que posee mayor precisión que los sistemas RFID se opta por crear un sistema de posicionamiento indoor basado en visión artificial y con el objetivo de reducir la cantidad de datos y computación para llevar acabo el posicionamiento se opta por realizar el mapeo y localización por separado siendo el método escogido para crear el mapa la AR con la finalidad de reducir de nuevo el coste computacional y lograr que los sistemas embarcados sean ligeros y de bajo consumo.

3 Visión artificial

Por definición se refiere *"La visión artificial es una disciplina científica que incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de producir información que pueda ser tratada por una máquina."*

Será necesario el uso de una cámara que sea adecuada para el proyecto. Debido al carácter de este, se establecen las características básicas que debe poseer.

- Alta resolución 1k (16Mp)
- Alto numero de FPS (30-60 FPS)
- Bajo peso (150 g)

Estos requisitos se deben a que debemos ser capaz de captar marcadores con suficiente precisión y que estas imágenes no salgan borrosas debido a la toma dinámica de imágenes y por ello necesitamos un número elevado de FPS por lo que teniendo todo esto en cuenta se buscan de entre todos los tipos de cámaras los que se aproximen mas a estas características.

3.1 Tipos de cámaras

Dependiendo de que son capaces de detectar en concreto el espectro de frecuencias que pueden percibir:

- Cámaras multiespectrales
- Cámaras térmicas
- Cámaras RGB

Las cámaras multiespectrales te permiten observar rangos ultravioleta e infrarojo muy útiles en el sector agrario, ya que te permiten analizar el estrés hídrico de las cosechas entre otros aspectos. Estas camaras por lo general tienen baja resolución y un numero bajo de FPS por lo que no cumplen los requisitos.

En cuanto a las cámaras térmicas permiten ver las diferentes temperaturas del entorno pero no las referencias visuales de marcadores, por lo que no es valido para este proyecto.

Por ultimo la cámaras RGB son las mas comunes y las que existen al alcance de cualquiera. Existen diferentes tipos de sensores pero los que poseen un mejor ratio Megapixeles/FPS son los que usan tecnología CMOS.

Los sensores CMOS (Complementary Metal Oxide Semiconductor) incluyen en su substrato, el área activa del píxel y el espacio necesario para el chip que se encuentra en el propio circuito. La ventaja principal de los sensores CMOS es su velocidad.

Teniendo en cuenta que el mayor distribuidor de estos sensores es SONY se procede a elegir uno de sus productos atendiendo a las prestaciones anteriores.

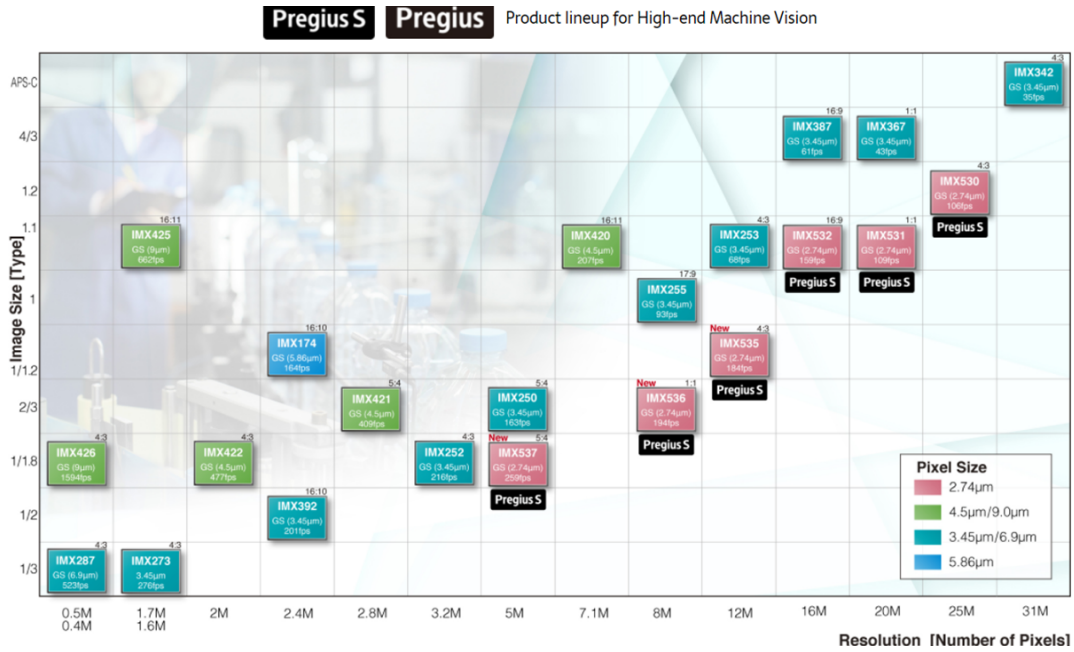

Figura 3.1 Comparativa de relaciones de aspecto con resolución de imagen.

Atendiendo a esta gráfica observamos que la linea pregius S es la q se adapta mejor, por lo que haciendo un análisis mas detallado de esta linea obtenemos el modelo mas adecuado para esta tarea.

New IMX571BLR/BQR-J		Rolling Shutter CMOS 26.1M, 1.8"	26.1 M	6252 x 4176	1.8	3 : 2	3.76	48	SLVS-EC	Monochrome RGB
IMX530-AAMJ/AAQJ		Global Shutter CMOS 24.5M, 1.2"	24.5 M	5328 x 4608	1.2	4 : 3	2.74	106	SLVS SLVS-EC	Monochrome RGB
IMX540-AAMJ/AAQJ		Global Shutter CMOS 24.5M, 1.2"	24.5 M	5328 x 4608	1.2	4 : 3	2.74	35	SLVS SLVS-EC	Monochrome RGB
IMX183CLK-J/CQJ-J		Rolling Shutter CMOS 20.4M, 1"	20.4 M	5544 x 3694	1	3 : 2	2.4	24	sub LVDS Serial	Monochrome RGB
IMX531-AAMJ/AAQJ		Global Shutter CMOS 20.3M, 1.1"	20.3 M	4512 x 4512	1.1	1 : 1	2.74	109	SLVS SLVS-EC	Monochrome RGB
IMX541-AAMJ/AAQJ		Global Shutter CMOS 20.3M, 1.1"	20.3 M	4512 x 4512	1.1	1 : 1	2.74	42	SLVS SLVS-EC	Monochrome RGB
IMX367LLA/LQA		Global Shutter CMOS 19.6M, 4/3"	19.6 M	4432 x 4436	4/3	1 : 1	3.45	43	SLVS SLVS-EC	Monochrome RGB
IMX387LLA/LQA		Global Shutter CMOS 16.8M, 4/3"	16.8 M	5472 x 3084	4/3	16 : 9	3.45	61	SLVS SLVS-EC	Monochrome RGB
IMX532-AAMJ/AAQJ		Global Shutter CMOS 16.1M, 1.1"	16.1 M	5328 x 3040	1.1	16 : 9	2.74	159	SLVS SLVS-EC	Monochrome RGB
IMX542-AAMJ/AAQJ		Global Shutter CMOS 16.1M, 1.1"	16.1 M	5328 x 3040	1.1	16 : 9	2.74	52	SLVS SLVS-EC	Monochrome RGB
New IMX535-AAMJ/AAQJ		Global Shutter CMOS 12.4M, 1/1.1"	12.4 M	4128 x 3008	1/1.1	4 : 3	2.74	184	SLVS SLVS-EC	Monochrome RGB
New IMX545-AAMJ/AAQJ		Global Shutter CMOS 12.4M, 1/1.1"	12.4 M	4128 x 3008	1/1.1	4 : 3	2.74	68	SLVS SLVS-EC	Monochrome RGB
New IMX565-AAMJ/AAQJ		Global Shutter CMOS 12.4M, 1/1.1"	12.4 M	4128 x 3008	1/1.1	4 : 3	2.74	42	MIPI CSI-2	Monochrome RGB

Figura 3.2 Sensores de la línea pregius.

Se aprecia dos modelos claramente superiores en FPS el modelo IMX532 y el IMX531, se escoge el IMX532 ya que se priorizan los FPS puesto que la calidad de imagen no es excesivamente inferior, con el fin de realizar imágenes menos borrosas al tomar fotos en movimiento.

Basic Drive Mode

Drive mode	Recommended number of recording pixels	Maximum frame rate [frame/s]	Output interface	ADC [bit]
All pixel	5320 (H) × 3032 (V) approx. 16.13 M pixels	45	SLVS 8 ch	8
		159	SLVS – EC 8 Lane	
		36	SLVS 8 ch	10
		152	SLVS – EC 8 Lane	
		32	SLVS 8 ch	12
		111	SLVS – EC 8 Lane	
Vertical / Horizontal 1/2 subsampling	2660 (H) × 1516 (V) approx. 4.03 M pixels	149	SLVS 8 ch	8
		308	SLVS – EC 8 Lane	
		120	SLVS 8 ch	10
		295	SLVS – EC 8 Lane	
		119	SLVS 8 ch	12
		216	SLVS – EC 8 Lane	

Figura 3.3 Características cámara IMX532.

Finalmente se escoge el modelo IMX532 porque posee una calidad aceptable de 16.1 Mp y alto numero de FPS (159 FPS). Este modelo es ampliamente usado en productos como cámaras deportivas de gran angular como son las GoPro.

3.2 Calibración

Todas las cámaras poseen distorsiones que deforman como se percibe realmente el espacio por ellos es necesario hacer correcciones.

Existen dos distorsiones típicas:

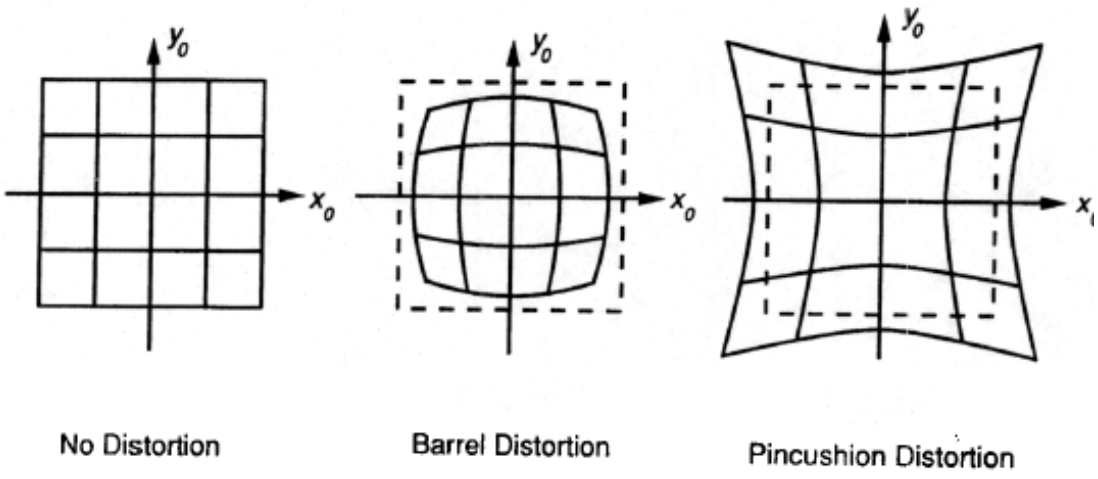

Figura 3.4 Distorsiones radiales.

Para corregir estas distorsiones se usa la librería OpenCV especializada en visión artificial y con un amplio contenido de funcionalidades que permiten hacer muchos tipos de análisis a las imágenes, entre ellos hay que destacar el metodo:
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
Que permite calibrar la imagen aunque para ello hay que preparar el entorno y crear un progrma para la calibración, el diagrama de flujo ilustra como proceder para realizar el algoritmo de calibración.

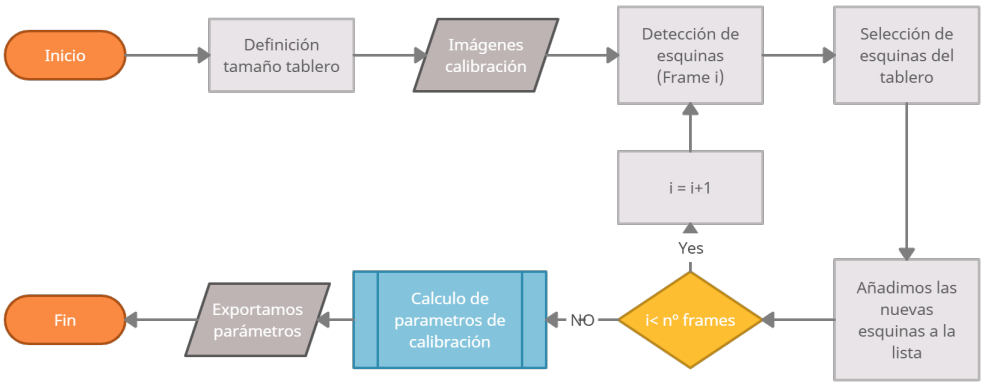

Figura 3.5 Diagrama de flujo del algoritmo de calibración.

1. Se define el tamaño del tablero que el programa utiliza para la calibración, en este caso se ha utilizado un tablero de 10x7 que tiene 9x6 esquinas interiores.

2. Se importan las imágenes de referencia tomadas con la cámara que se va a calibrar, mínimo 10. Se selecciona la primera imagen.

3. Tras esto, se detectan las esquinas del tablero en la imagen y se seleccionan.

4. Se añaden estas posiciones de las esquinas del tablero a una lista de datos y se regresa al paso 3 con la siguiente imagen de calibración. Este bucle se repite hasta que se han tratado todas las imágenes de calibración.

5. Se añaden estas posiciones de las esquinas del tablero a una lista de datos y se regresa al paso 3 con la siguiente imagen de calibración. Este bucle se repite hasta que se han tratado todas las imágenes de calibración.

6. Se realiza una estimación de los parámetros de calibración utilizando los datos almacenados en la lista de posiciones de las esquinas, ajustando los resultados con las funciones de las distorsiones y de la cámara.

7. Se exportan los parámetros obtenidos para utilizarse en el tratamiento de imágenes posterior.

Este tablero de ajedrez se imprime y se coloca en una superficie plana, a la cual se tomaran imágenes desde diferentes perspectivas.

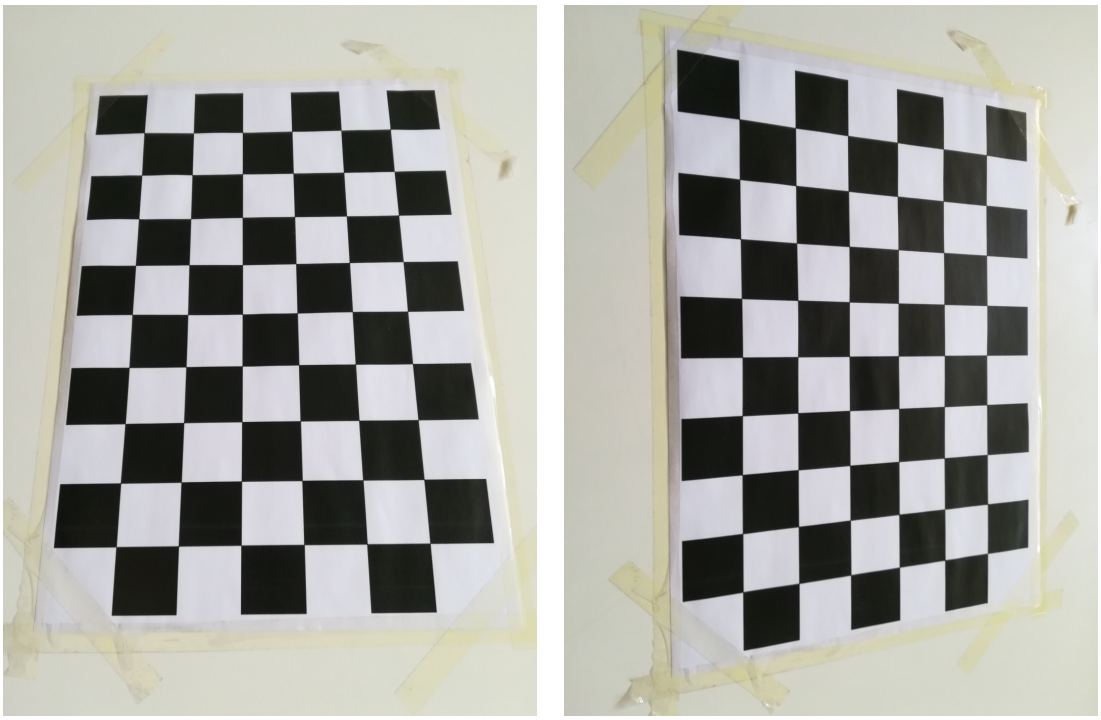

Figura 3.6 Imágenes de calibración.

Como se ha descrito en los puntos anteriores se extraen las esquinas con la finalidad de determinar las desviaciones radiales esta detección se puede observar en la figura inferior y sucede en el punto 3 descrito.

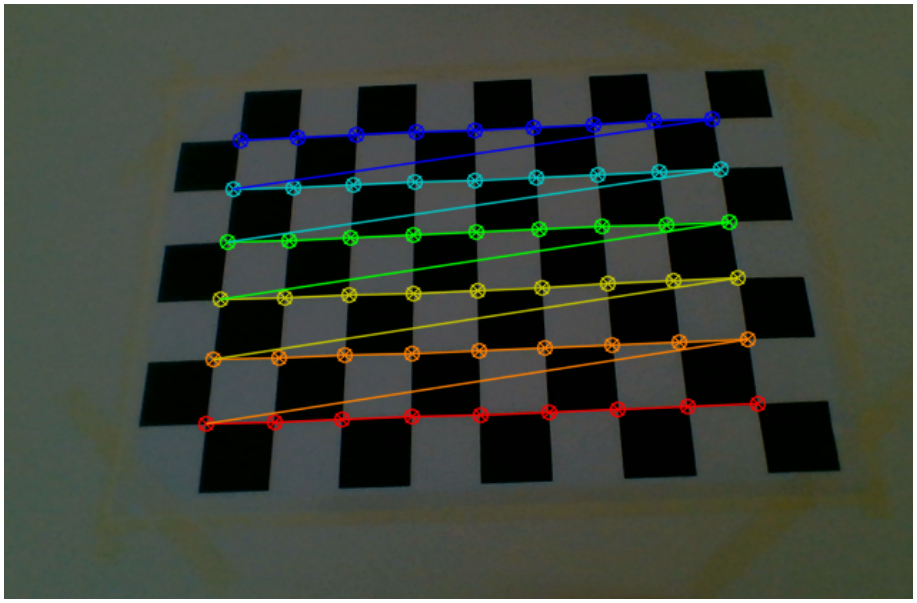

Figura 3.7 Esquinas interiores detectadas.

El algoritmo de calibración se encuentra en el anexo. Como se puede apreciar los programas están realizados mediante lenguaje python puesto que permite realizar un desarrollo mas rápido frente a su alternativa C/C++ ya que es un lenguaje de mas alto nivel y debido a que es un lenguaje interpretado a diferencia de C/C++ lo que permite ejecutar el programa hasta el punto donde se encuentra el error permite una depuración mas rápida aunque a nivel de rapidez de ejecución sea significativamente inferior a lenguajes precompilados.

3.3 Realidad aumentada

Este es un concepto que aparece en 1990 la cual tiene como fin visualizar el "mundo real" a través de un dispositivo tecnológico integrando información gráfica añadida a este.

Este tecnología es importante en muchos sectores como los puede ser la medicina, el cultural, el industrial y en juegos. Nos centraremos en una aplicación que se caracteriza por el uso de una serie de referencias visuales llamadas marcadores, usados como puntos de referencia de las imágenes para añadir información gráfica.

Este caso particular se popularizó mucho tanto en la educación como en el desarrollo de juegos basado en esto. Gracias a este auge surgieron diferentes programas que ofrecían estas características.

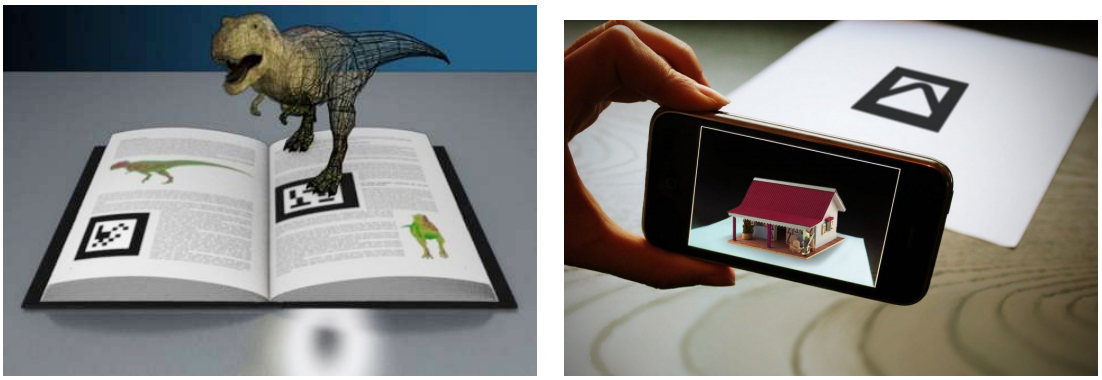

Figura 3.8 Ejemplos de realidad aumentada.

Como se puede apreciar se usan estos marcadores como referencias para superponer imágenes sobre los marcadores para que sea coherente con el entorno.

Debido a este interés y la versatilidad de esta tecnología un grupo de investigación de la universidad de Córdoba desarrollan una librería escrita en Python y C++ llamada AruCo que permite reconocer marcadores y darnos información sobre ellos.

3.3.1 Marcadores AruCo

Estas son las referencias visuales que usan los algoritmos desarrollados por la UCO existen dos tipos el marcador tradicional y el encapsulado, izquierda y derecha respectivamente. El encapsulado da mas precisión en la detección de las esquinas pero es mas inestable pues no siempre es capaz de detectar dichas esquinas, por no decir que el tiempo de computación es mayor, sin embargo el tradicional tiene una menor precisión aunque bastante cercana al encapsulado y con tiempos de computación significativamente inferior y mayor estabilidad. Por lo que estos primeros serán los usados posteriormente.

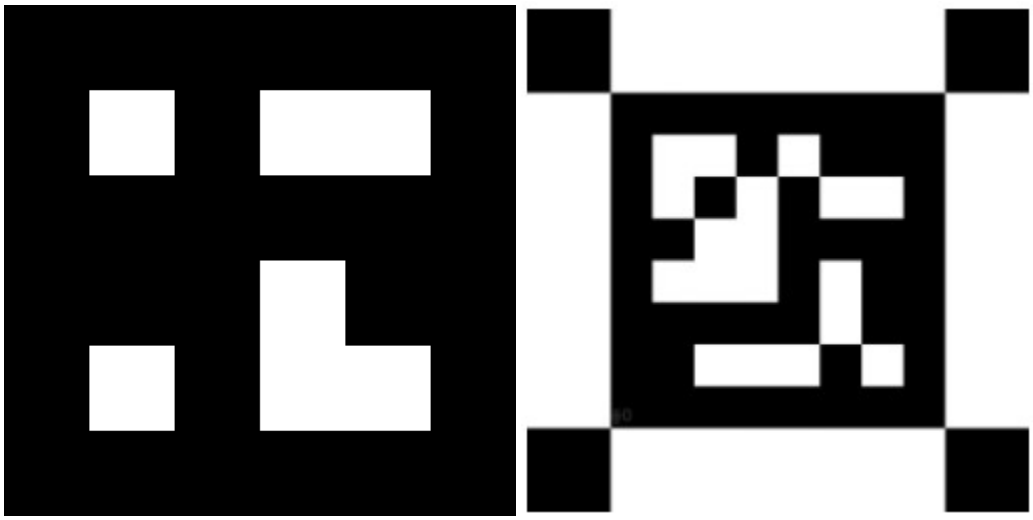

Figura 3.9 Ejemplos de marcador: 4x4 a la izquierda y marcador encapsulado 6x6 a la derecha.

Es necesario destacar las variaciones propias de cada marcador, el patrón interior del marcador determina la ID de cada uno con un numero por lo que dependiendo de cuantas combinaciones se puedan hacer en su interior poseerá mayor numero de ID diferentes, por lo que aparecen marcadores 4x4 5x5 ... 10x10 y todo depende del mallado interno de los marcadores. Debido a que nuestra aplicación es a pequeña escala con el marcador más básico servirá, ese es el caso del 4x4 un ejemplo de este se encuentra en la figura anterior, el marcador izquierdo.

3.3.2 Metodología de detección

Se basa en un preprocesamiento de la imagen y una posterior selección y descarte de posibles marcadores.

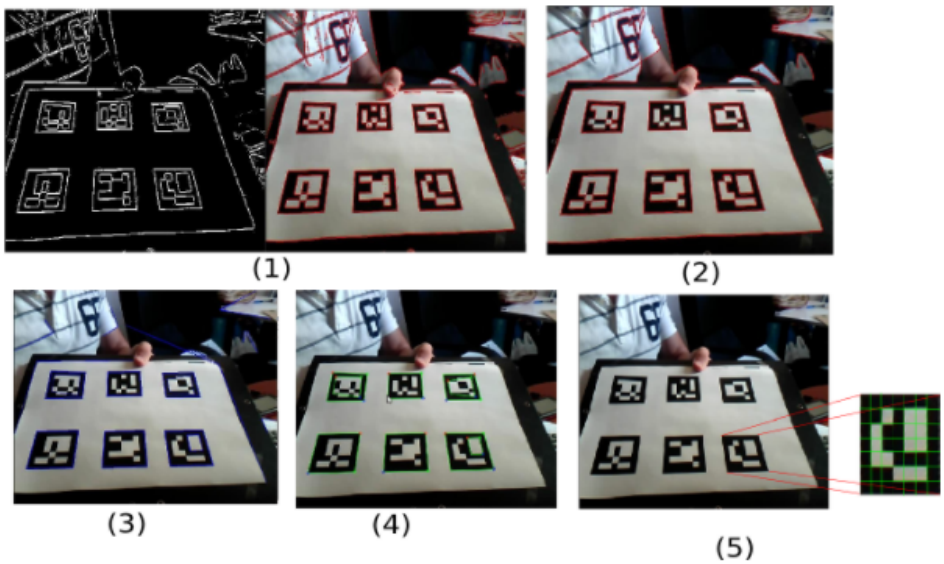

Figura 3.10 Proceso de detección.

1. Mediante el uso de algoritmos de detección de bordes como Canny, extraemos una nueva imagen con estos bordes.
2. Eliminamos aquellos bordes con bajo numero de puntos.
3. Descartamos todos los bordes que no formen rectángulos quedándonos así muy cerca de acotar el marcador.

4. Eliminamos los rectángulos que se encuentran en el interior de otros. Esto es necesario pues normalmente dentro del marcador detectas rectángulos que no son marcadores.
5. Creamos una celda de malla nxn donde n esta definido por el tipo de marcador aruco que usamos como mencionamos en la sección de marcadores.

Por ultimo creamos una celda de malla nxn donde n esta definido por el tipo de marcador aruco que usamos como mencionamos en la introducción.

Para todo esto usamos las funciones implementadas por el modulo Aruco
"marker_corners, ids, rejectedImgPoints = aruco.detectMarkers(frame_gray, aruco_dict, parameters=arucoParameters)"

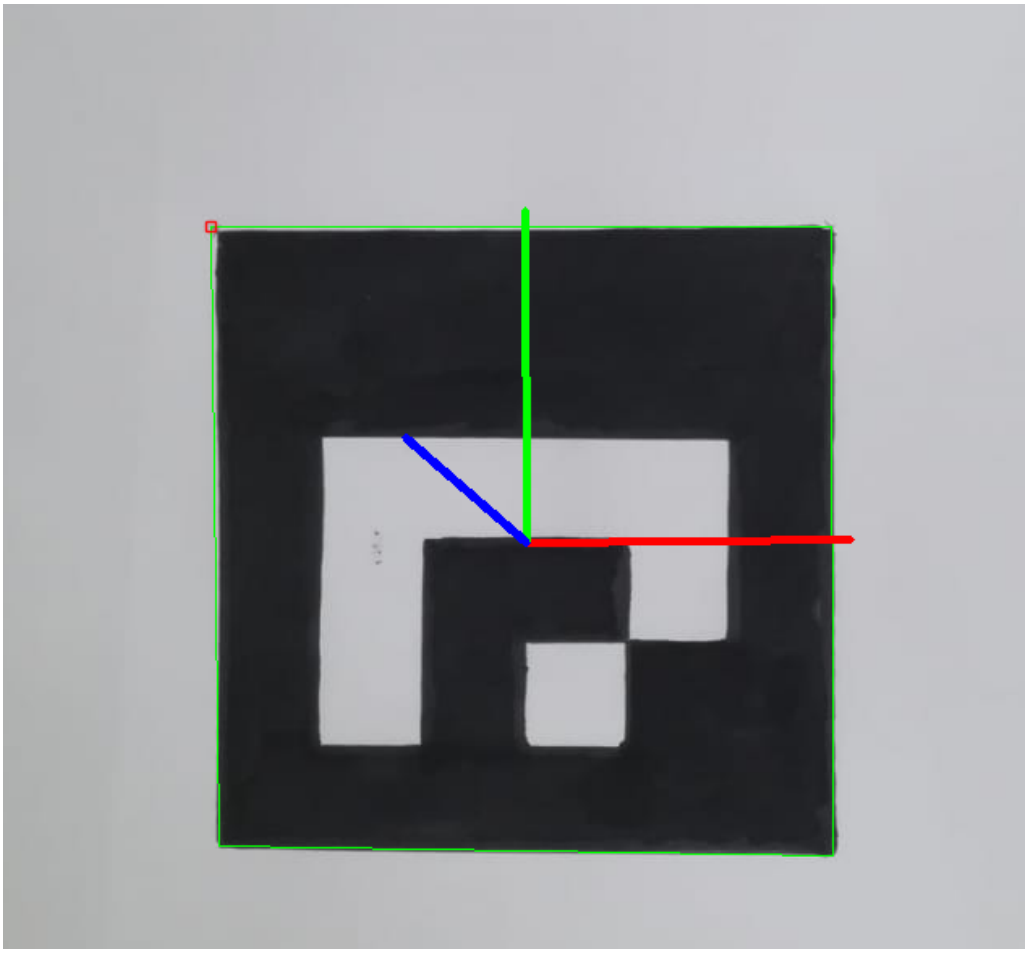

Figura 3.11 Ejes marcador.

Este es un ejemplo de la detección de un marcador del diccionario de 4x4 en el cual se han superpuestos los ejes calculados del marcador.

4 Algoritmia de mapeado

Con el objetivo de realizar un programa funcional que genere una nube de puntos que sitúen todos los marcadores de una instalación ha de seguir un riguroso proceso de tratamiento de información. El diagrama de flujo de este programa de forma global es el siguiente.

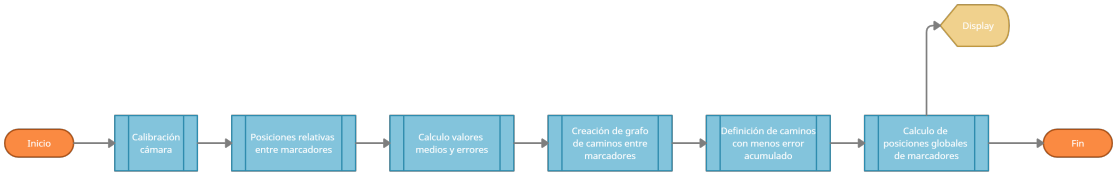

Figura 4.1 Diagrama de flujo algoritmo de mapeado.

En rasgos generales el funcionamiento es sencillo, crea el mapa una vez conocidas las posiciones relativas entre marcadores, es decir, si establecemos la posición del primer marcador, conociendo la posición relativa con otros dos, estos quedan localizados y, a su vez estarán relacionados con otros, por lo que acumulando estas distancias entre marcadores seremos capaces de reconstruir todo el panorama, aunque para ello de forma interna tengamos que aplicar conceptos mas complejos, en otras palabras se irán estableciendo relaciones de posición entre marcadores como si de un puzzle se tratara.

El primer paso ya se encuentra explicado en el capitulo de visión artificial. Por lo que solo se explicaran los siguientes procesos.

4.1 Recopilación de datos multimedia

Es un punto importante pues el hecho de que los datos multimedia recopilados sean de calidad supondrá un mejor desempeño del algoritmo. Se tomaran mínimo 10 imágenes completas del tablero de calibración lo mas cerca posible, y en posiciones diferentes pues fotos similares sería redundantes a la hora de la calibración, tomando imágenes que puedan ilustrar correctamente todas las distorsiones de la cámara.

El vídeo de las instalaciones obviamente también es muy importante por lo que se tomara en condiciones lumínicas óptimas intentando que los marcadores se vean los mas cerca posible pues conforme la cámara se aleja a ellos la precisión es menor, de la misma forma ocurre con el ángulo entre la cámara y el marcador, lo mejor es tomar una imagen lo mas paralela posible y por ultimo pero no menos importante es que mínimo deben grabarse dos marcadores a la vez pues si solo se graba uno no se podrán generar posiciones relativas

4.2 Posiciones relativas

El objetivo de este es crear una lista con la información correspondiente a las distancias entre marcadores.

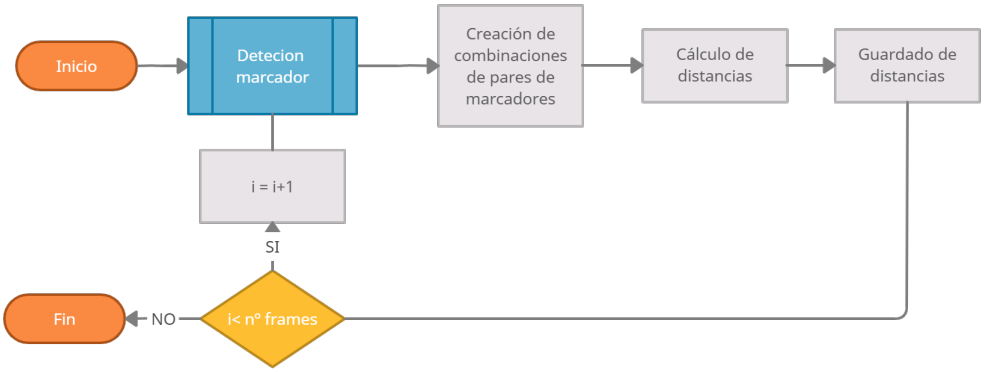

Figura 4.2 Diagrama de flujo de posiciones relativas .

Por lo que será necesario conocer las variables tvec y rvec que son vectores de posición y rotación con respecto a ejes cámara. Estos ejes cámara se definen como eje x la horizontal de la cámara, y la vertical apuntando hacia abajo y z perpendicular a la cámara saliendo de ella.

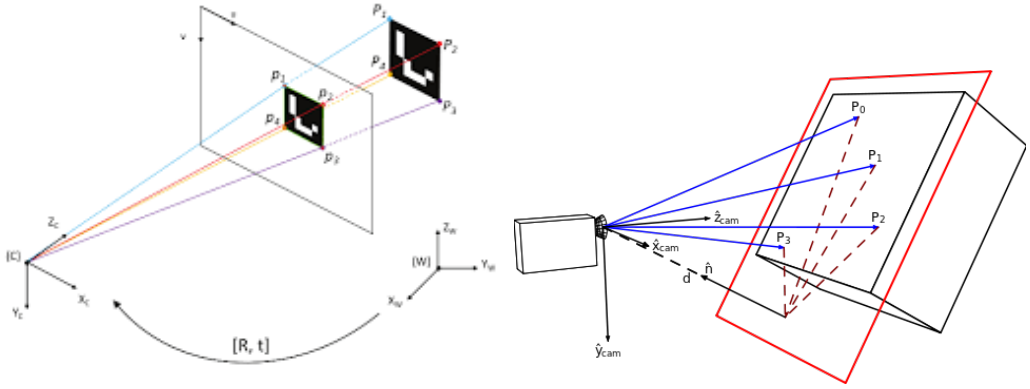

Figura 4.3 Convenio de ejes cámara Aruco.

Para generar esa lista de distancias previamente tenemos que calcularlas pero ya que necesitamos que esten medidas en ejes marcador pues ejes camara van a ir cambiando conforme se realicen las fotos de los marcadores debemos hacer cambios de base.

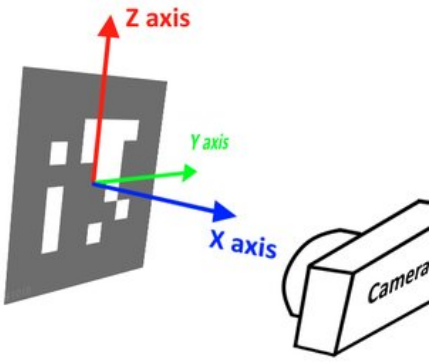

Figura 4.4 Convenio de ejes marcador Aruco.

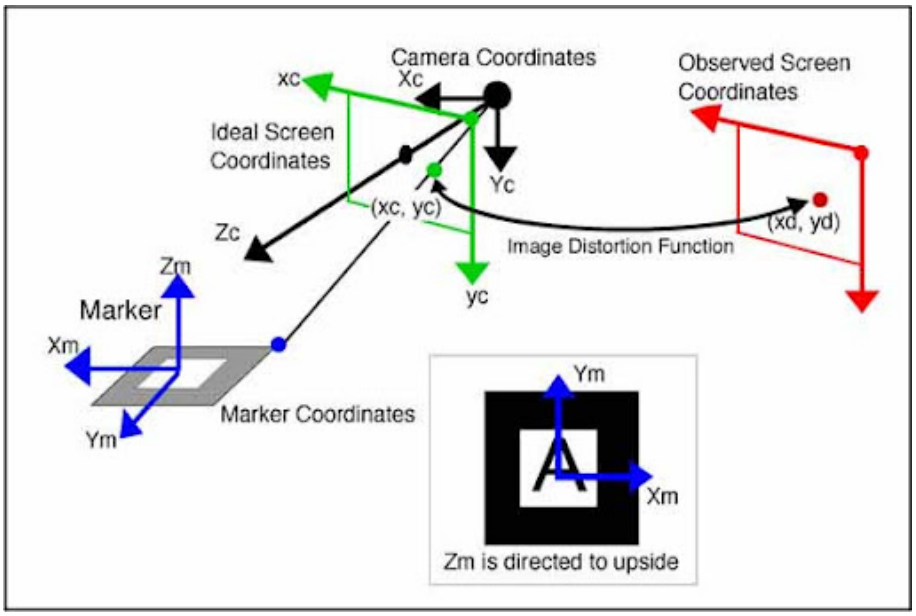

Figura 4.5 Convenio de ejes marcador Aruco.

Con el objetivo de dejar claro los cambios de base y que vectores estamos calculando se representa el esquema inferior.

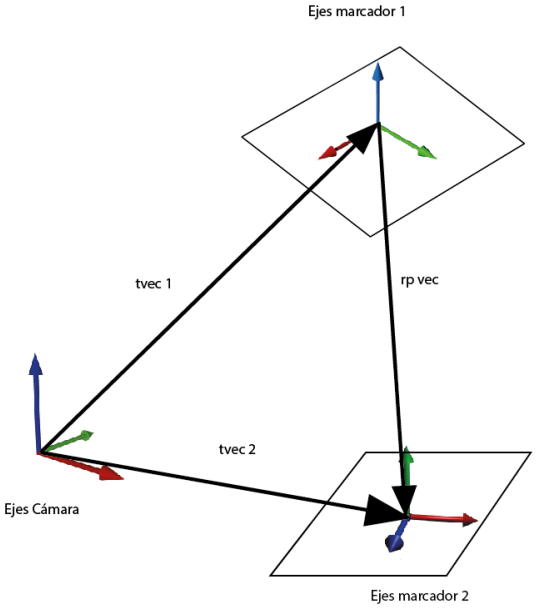

Figura 4.6 Ejes marcador y cámara .

Haciendo caso de la figura anterior para generar el vector rp_{vec} que hace referencia a la distancia entre marcadores en ejes marcador 1 se calcularía como,

$$rp_{vec} = (tvec_2 - tvec_1)R_{01}$$

Siendo R_{01} la matriz de rotación de los ejes "0" (ejes cámara) a "1" (ejes marcador 1).

Esta información se guarda en una tabla de datos del tipo DataFrame típico en BigData y manejo de bases de datos para aligerar el tratamiento de datos, el procedimiento se basa en escoger todas las combinaciones

de pares de marcadores, calcular su distancia y almacenarla por frame como queda representado en diagrama de flujo del principio de la sección.

El uso de DataFrames es conveniente cuando se manejan grandes cantidades de datos, pues la librería pandas de python que manejan estas estructuras implementan funciones para hacer modificaciones, agrupaciones, métodos estadísticos y otros métodos de forma muy eficiente, mucho mas que en cualquier estructura tipo array y usando bucles "for" para hacer dichas modificaciones.

Al realizar pruebas de detección de marcadores observamos que el algoritmo de detección tiene serios problemas para calcular el eje Z de los marcadores, que aunque para un marcador el resultado es aceptable, cuando se genera el mapa se generan muchos errores por lo que se propone un método para solucionarlo.

4.2.1 Corrección del eje Z del marcador

Debido a altos números de condición en las matrices con las cuales se calcula la orientación del eje Z, este suele poseer mucho error. Analizando en mas detalle los resultados se aprecia que el calculo de la posición de los marcadores respecto a la cámara son muy buenos, por tanto como en la mayoría de imágenes se detectaran 3 o mas marcadores podremos determinar el plano que forman entre sí, claro está que estos marcadores tendrán que estar sobre una misma superficie dando lugar así a un grupo de marcadores con la misma normal y que la normal al plano que forman este grupo de marcadores.

En resumen se detectan los marcadores de una imagen, se agrupan por marcadores que se encuentren en la misma superficie y por ultimo si los grupos son superiores a 2 se calcula la normal del plano que forman sustituyendo la normal de todos los marcadores agrupados por la del plano calculado.

Una vez explicado en lineas generales pasamos a explicar el proceso en mayor profundidad y para ello usamos de refuerzo el siguiente diagrama.

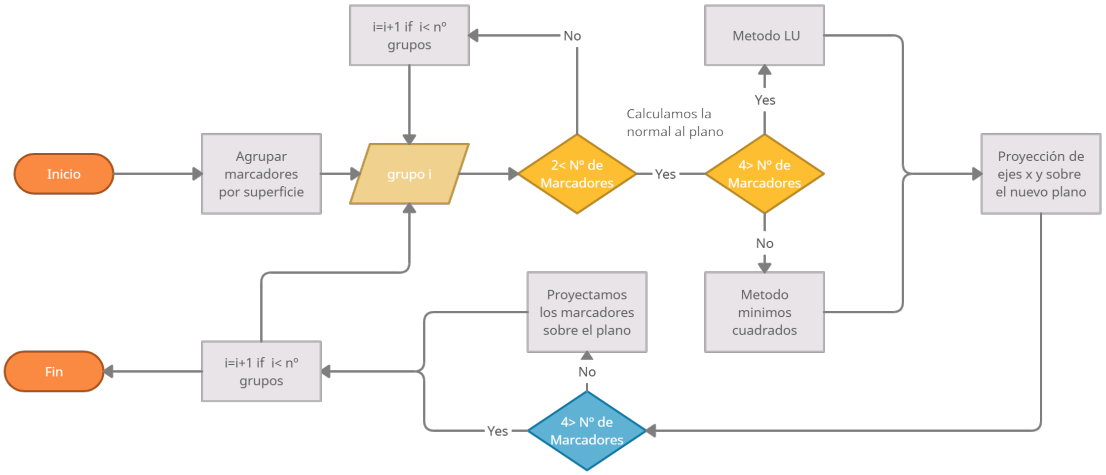

Figura 4.7 Diagrama de flujo de corrección del eje Z .

1. Se agrupan marcadores por pertenencia a la misma superficie, es decir que todos los marcadores compartan el mismo vector normal, para ello si el eje z entre marcadores forman un ángulo inferior a 10º se consideran en el mismo plano.

2. Por cada grupo con mas de 2 marcadores se calcula el vector normal formado por los marcadores, si no cumple esto salen fuera de este proceso quedando sin corregir su normal.

3. Elegimos el método mas apropiado para el calculo de la normal. Si solo tenemos 3 marcadores existe una única solución posible, por lo que se usan métodos como el LU para resolver el sistema de ecuaciones. Sin embargo si tenemos un sistema sobredefinido necesitaremos resolver por mínimos cuadrados.

4. Para comprobar que el resultado es preciso se calcula el numero de condición de las matrices que resuelven estos sistemas y se impone el limite de 100 lo que significa que un error de medición se multiplicara por 100 en el resultado.
5. Una vez modificado el eje Z hay que modificar convenientemente los ejes x e y para que sigan siendo ortonormales para ello se reproyectan según el eje Z.
6. Si la solución se ha dado por mínimos cuadrados significa que los marcadores no tienen porque pertenecer al plano solución por lo que se insertan en este encontrando la solución de mínimos cuadrados.

Y con esto quedan rectificadas las matrices de rotación de los marcadores obteniendo mayor precisión para el calculo de distancias relativas.

4.3 Grafo

La teoría de grafos es un área de las matemáticas que se define como *"conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto."*

Es decir te permite establecer relaciones entre diferentes objetos, en este caso de estudio son los marcadores y las relaciones son aquellas en las que conocemos su distancias relativas.

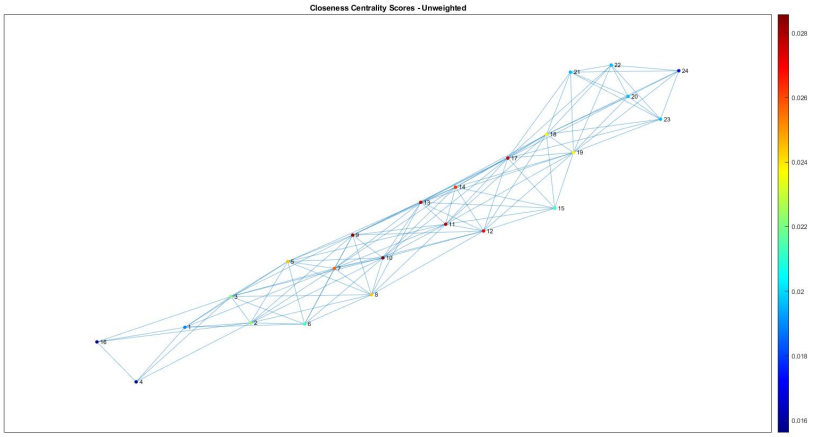

Figura 4.8 Ejemplo práctico de grafo de una operación de mapeo.

Aquí se muestra un grafo que nos muestra las relaciones entre marcadores y como se puede observar todos no están conectados, con este diagrama se puede ver mas claramente como se puede llegar a reconstruir la nube de puntos y es mediante las sucesivas interrelaciones de marcadores. Claramente se puede deducir como el ultimo marcador va a acumular bastante error puesto que para calcular su posición inicial hemos tenido que calcular el resto previamente.

Como consecuencia se plantea como reducir al máximo estos errores acumulados, y esta es la razón de crear un grafo. Si ponderamos los caminos de los grafos con los errores de distancia entre marcadores, recorrer el camino mas corto entro dos nodos significará generar la posición de los marcadores con menor error acumulado. Esto surge de que se puede fijar la posición de un determinado marcador a través de relaciones de marcadores muy dispares, por lo que pudiendo elegir se escoge la forma mas precisa.

Estos errores relativos son calculados en el modulo de posiciones relativas mediante el calculo de la desviación típica de todas las medidas. Poseemos una gran cantidad de mediciones repetidas puesto que en un vídeo con alto numero de FPS imágenes muy próximas en el tiempo serán prácticamente idénticas por lo que podemos obtener la incertidumbre de la medida con bastante exactitud.

Para el calculo de el camino mas corto entre dos nodos de un grafo se ha usado el algoritmo de Floyd-Warshall que tiene como objetivo crear la matriz una matriz de predecesores.

Las matrices de predecesores en teoría de grafos son por definición *"La matriz de predecesores de un grafo nos indica el camino más corto de un nodo a otro para todos los nodos de un grafo."*

Se hace uso del algoritmo ya citado que se basa en comparar todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3 comparaciones (esto es notable considerando que puede haber hasta V^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Una vez presentado el motivo de su creación se presenta el diagrama de flujo de esta sección de codigo del mapeado.

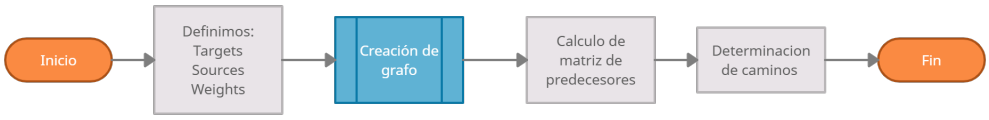

Figura 4.9 Diagrama de flujo grafos.

En resumen definimos los parámetros del grafo Targets, Sources y weights.

- Targets: Nodos de partida
- Sources: Nodos destino
- Weights: Pesos de los caminos entres Tragets y Sources

Como las relaciones entre marcadores son bidireccionales estos pares de relaciones tiene que aparecer como Targets y como Sources indistintamente.

Una vez creada la matriz de predecesores hay que generar una matriz en la cual cada fila será el camino mas corto a seguir para alcanzar el nodo de la fila es decir, si observamos la fila 5 de la matriz haremos referencia al camino mas corto entre el marcador 5 y el 0.

Para generar esta matriz tendremos que recorrer la matriz de predecesores correctamente y para ello en nuestro caso solo nos interesa la primera fila de la matriz de predecesores pues es el que nos proporciona el camino mas corto entre cualquier marcador y el 0 que es el que se toma de forma predeterminado como la referencia.

En resumen el valor del elemento Mp(0,4) nos dará cual es nodo anterior al 4 perteneciente al camino mas corto. por ejemplo si nuestra primera fila es:

$$MP = \begin{bmatrix} -1 & 4 & 1 & 0 & 0 \end{bmatrix}$$

Denominamos MP a la matriz de predecesores.

Para ir del nodo 1 al 0 que camino deberíamos recorrer pues el primer nodo efectivamente es el 1 luego MP(0,1) = 4 por lo que el siguiente es el 4 y por tanto MP(0,4)=0 y llego al ultimo nodo el 0 quedando el camino como 0 - 4 - 1 como camino mas corto.

4.4 Distancias globales entre marcadores

Para calcular las distancias globales usamos un sistema de actualización de matrices de rotación y seguimiento de caminos óptimos.

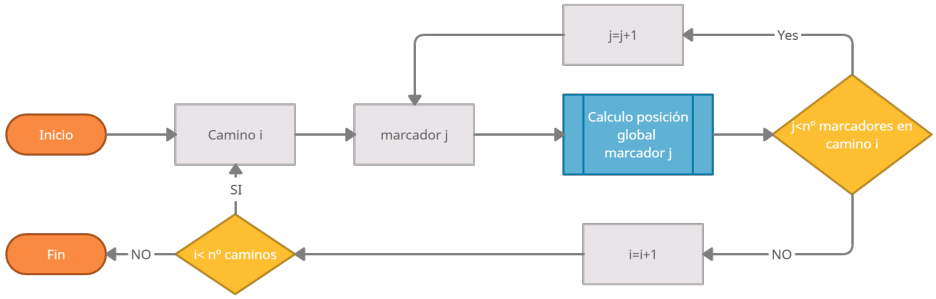

Figura 4.10 Diagrama de flujo de posiciones globales.

Como se ve en el diagrama de flujo se recorren todos los caminos y se pasan por todos los marcadores. Y para el calculo de la posición global con respecto a cada marcador se procede siguiendo estos pasos:

1. **Actualización de la matriz de rotación:** Ya que de cada marcador conocemos su distancia relativa con el anterior, se calculan sumando distancias, pero estas tienen que estar medidas en los ejes de referencia por tanto, $R_{j,0}R_{j-1,j} = R_{j-1,0}$.
2. **Calculo de la nueva posición:** Para ello $x_{j0} = x_{j-1,0} + R_{j-1,0}xr_{j,j-1}$ la notación de índices se basa en que el primer subíndice es la posición o matriz de rotación del marcador y el segundo en los ejes que están medidos siendo el 0 el de referencia y xr se refiere a la posición relativa de j con j-1 en ejes j-1.

Representación gráfica

Para la representación se han usado módulos de python como mpltoolkits y matplotlib para representar una nube de puntos aunque también existe la posibilidad de extraer estos datos como un fichero .mat con el objetivo de representarlo en matlab por comodidad se ha usado python.

5 Interfaz gráfica

Una vez realizado el mapa he implementando un sencillo algoritmo que híbrida los métodos de localización AoA y ToA podemos localizar el UAV dentro de nuestras instalaciones. Con el objetivo de visualizar esta información se crea una interfaz de usuario que posee un horizonte artificial la imagen que recibiría por la cámara del UAV así como una brújula y la posición absoluta con respecto al marcador referencia.

Para crear esta interfaz usaremos de nuevo el lenguaje python para dar consistencia a todos nuestros algoritmos, en concreto usaremos la librería de python llamada pygame, que se usa para crear sencillos juegos bidimensionales, por lo que para crear lo anteriormente enumerado es suficiente. Este programa funciona con la sucesiva superposición de imágenes por lo que si esta superposición es rápida entorno a un 30 - 60 FPS dispondríamos de un vídeo.

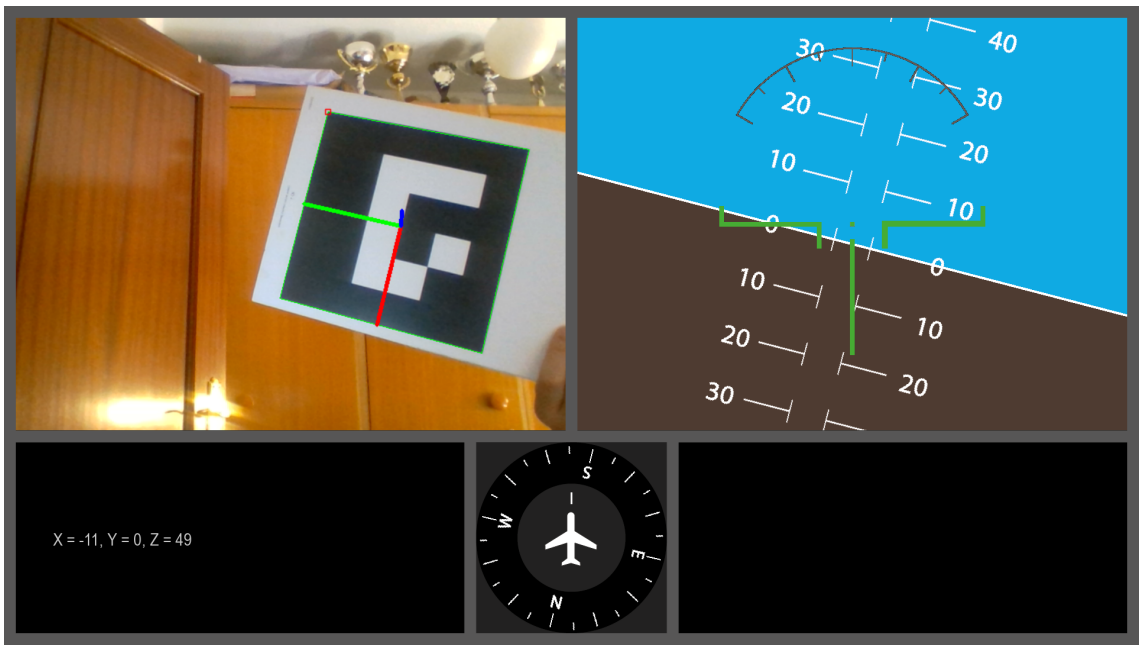

Figura 5.1 Interfaz gráfica.

Como se puede apreciar la pantalla esta separada en diferentes cuadrantes, el izquierdo es para la señal de vídeo el derecho para el horizonte artificial y en la parte inferior de la pantalla se muestra la información adicional como la brújula y la distancia entre cámara y marcador en ejes cámara.

5.1 Horizonte artificial

El horizonte artificial es un instrumento muy importante en las aeronaves pues muestra al piloto la actitud de esta indicando si esta nivelado y volando de forma horizontal. Con los sistemas digitales se sustituyen Aquellos analógicos por pantallas que muestran los datos recibidos directamente de dispositivos como el AHRS.

En este caso esa información la extrae de los marcadores con los que se ha creado el mapeado de las instalaciones. En este caso de prueba se ha utilizado un solo marcador que funciona como el de referencia por simplicidad. De la matriz de rotación del marcador se extraen los ángulos de euler para transformarlos en Yaw, Pitch y Roll de la cámara.

5.2 Horizonte artificial

Como ya comentamos ésta interfaz se genera superponiendo diferentes imágenes en este caso es solo una a la cual se le superpone la imagen principal de la interfaz.

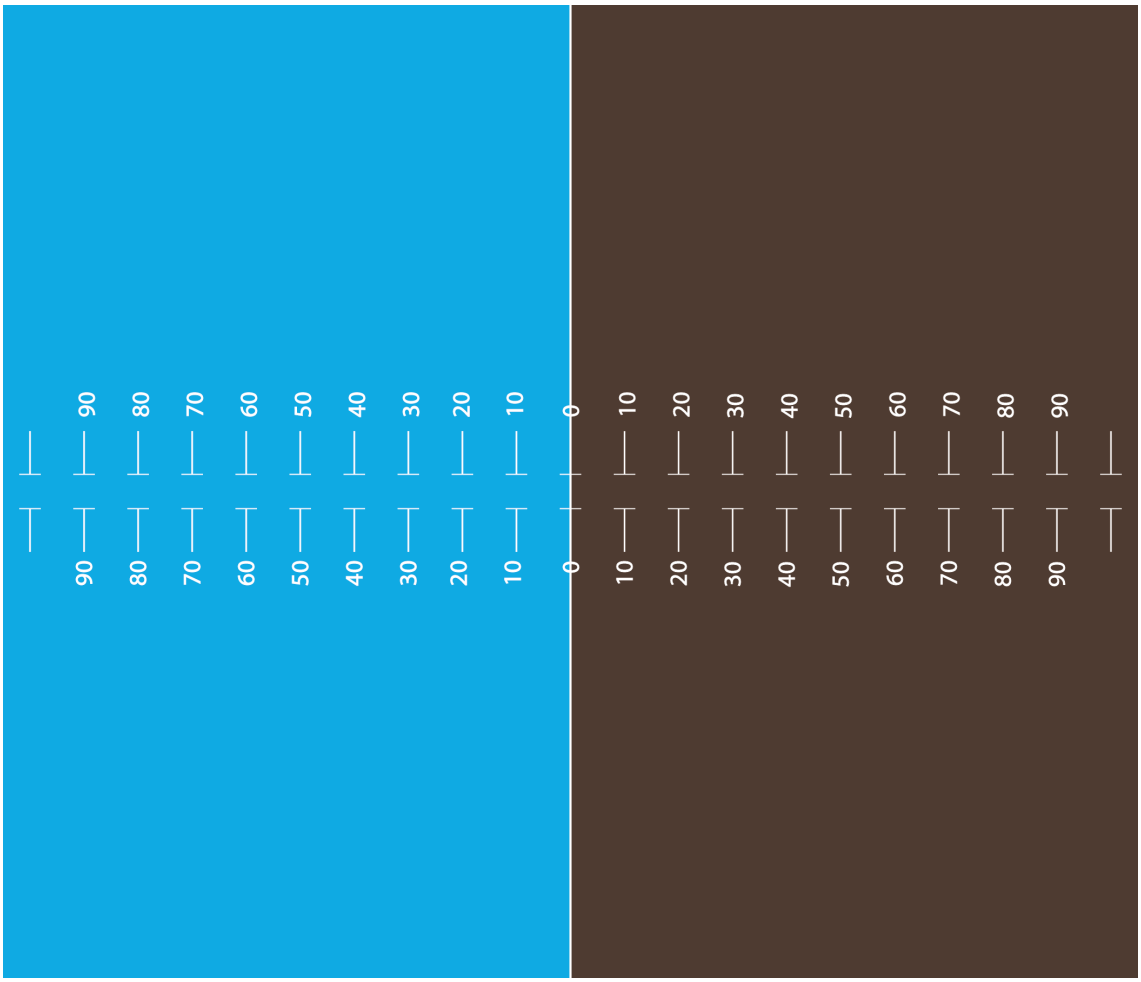

Figura 5.2 Cielo del HA.

Esta imagen es rotada y trasladada según roll y pitch, si el UAV posee 20º de pitch significa que las marcas de 20º sobre es el sector azul deben de encontrarse en el centro y si el roll es diferente, por ejemplo de 10º la imagen es rotada hacia la derecha dando lugar así al HA (horizonte artificial). Posteriormente se superpone la siguiente imagen dando por concluido el sector del HA.

Figura 5.3 Frame Principal.

Es importante aclarar que todas las partes negras de las imágenes funcionan como un canal alpha, es decir se comportan de forma transparente esta es la razón por la cual en el horizonte artificial se superponen las marcas de color gris y verde sobre el cielo del HA y no las partes en negro.

5.3 Brújula y coordenadas

Con la información del marcador extraemos la distancia a este y la mostramos en uno de los cuadrantes destinados a ello mientras que con el valor de Yaw que no se ha usado aun lo incorporamos en la brújula con el único objetivo de comprobar su correcto funcionamiento.

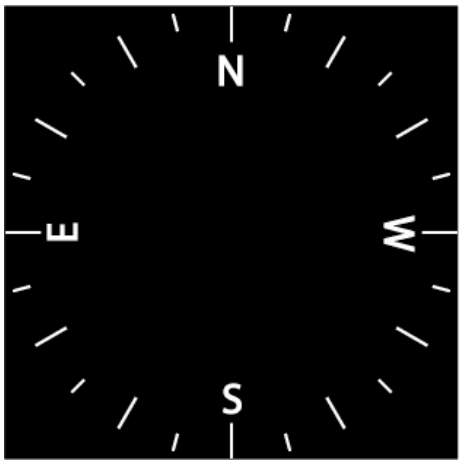

Figura 5.4 Brújula.

De nuevo esta imagen es rotada según Yaw para proporcionar el movimiento necesario para la comprobación de su correcto funcionamiento.

5.4 Multiprocesing

Esta interfaz gráfica se basa en los conceptos de simultaneidad de procesos esto quiere decir que existen 3 procesos principales:

- Interfaz gráfica
- Modulo de calculo de actitud y distancias
- Modulo administrador de memoria compartida y procesos

Esto significa que en el mismo instante de tiempo se están ejecutando 3 programas a la vez, cada uno en un núcleo diferente de nuestro procesador. Esto tiene un claro inconveniente, y es el hecho de que al ejecutarse el programa en núcleos diferentes las variables que poseen la información de posición y actitud al igual que de vídeo no pueden ser usadas por los 3 programas como en cualquier otro programa se haría.

Aparece el concepto de memoria compartida, este es un espacio reservado en nuestro ordenador usado para almacenar información a la que pueden acceder y modificar los diferentes procesos, por lo que estableciendo el tamaño de las variables a guardar, su tipo y guardando la localización de la misma, de una forma similar a trabajar con punteros en C, se puede acceder a esta información a través de los diferentes programas.

La estructura por tanto del programa es la siguiente:

1. Creación de los espacios con memoria compartida. Vídeo, ángulos y posición.

2. Definición del proceso de detección y calculo de posiciones y ángulos de la cámara.

3. Definición del proceso de actualización de frames de la interfaz.

4. Inicialización sucesiva de los dos procesos.

Por lo que para ambos procesos a su vez será necesario conectar los espacios de memoria compartida y una vez el programa finalice se borran los espacios de memoria creados y se desconectan de los procesos.

6 Resultados numéricos

Para probar nuestro algoritmo, hemos dispuesto una serie de situaciones a pequeña escala en la que conocemos las distancias entre marcadores, se realiza un vídeo de unos 3 segundos, se crean los frames asociados y se ejecuta el programa de mapeo. Pero antes de esto se han realizado pruebas sobre el propio programa de Aruco para comprobar con que fidelidad detecta los marcadores.

6.1 Errores de detección

Se han realizado dos experimentos el primero comprueba la dependencia del error en la medición con la distancia a la que se encuentra y el segundo permite conocer la desviación del calculo del eje Z del marcador.

6.1.1 Error en función de la distancia

Para esta prueba hemos usado un marcador de 18x18 cm del diccionario Aruco 4x4 y se han realizado medidas cada cierta distancia.

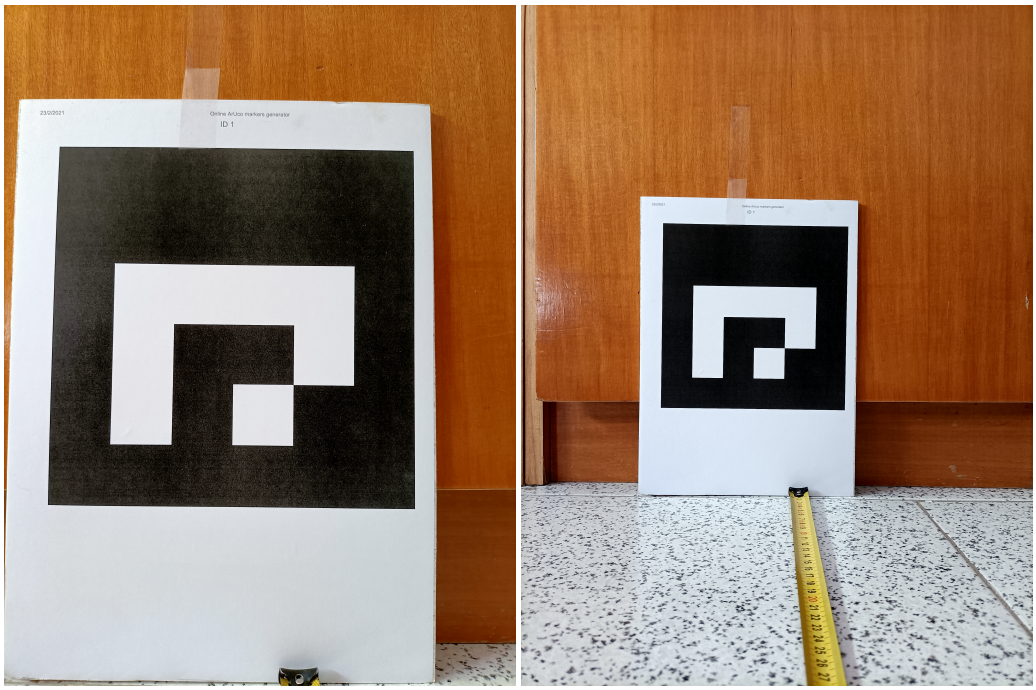

Figura 6.1 Prueba de errores en distancia.

Tras la toma de datos, apreciamos que la distribución de error acumulada es logarítmica con ajuste aceptable de $R = 0.9$ por lo que queda demostrado que al reconocer marcadores el error obtenido es mayor cuanto mas

alejado el marcador se encuentre, teniendo esto en cuenta lo ideal será tomar imágenes de los marcadores lo mas cerca posible.

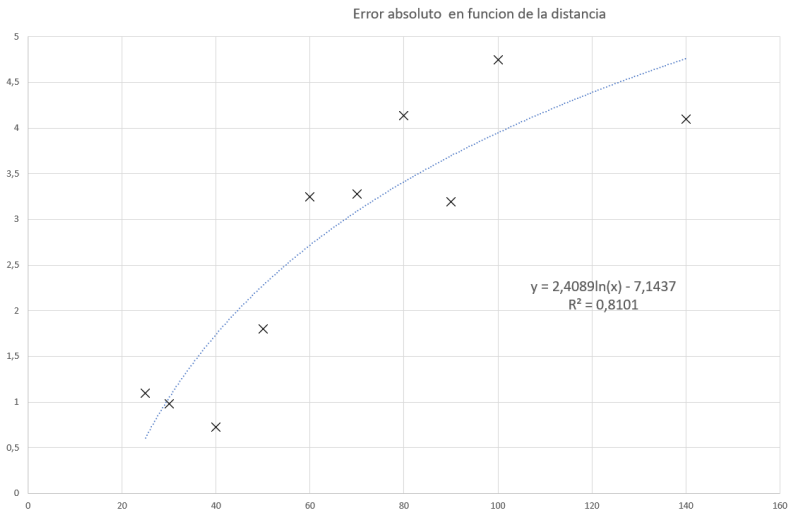

Figura 6.2 Gráfico errores en función de la distancia de medición.

6.1.2 Error en función del ángulo de observación

Al igual que en el caso anterior se usa el mismo marcador la diferencia es el modo en el que se toman las medidas en este caso se utiliza una plantilla en la que disponer la cámara y realizar las fotos en los ángulos correspondientes.

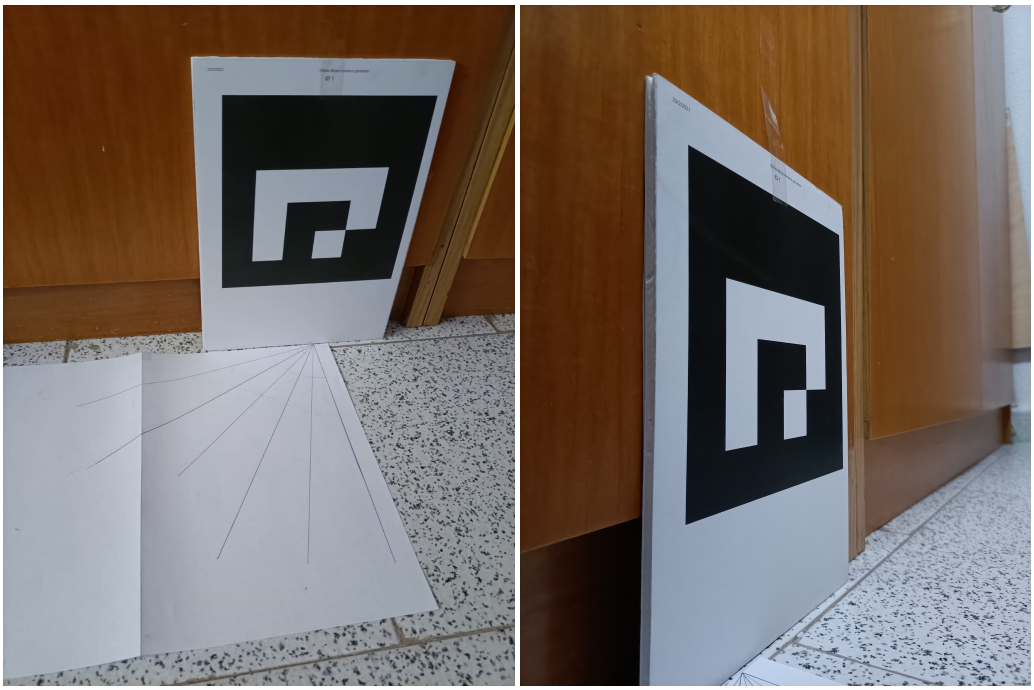

Figura 6.3 Prueba de errores en ángulo.

y como resultado obtenemos la siguiente distribución la cual no se puede ajustar a simple vista con alguna función, por lo que la conclusión a la que llegamos es que el calculo del eje Z es bastante inestable a ángulos mayores de 15º puesto q aparece un incremento importante en el error, siendo imposible su detección para ángulos mayores de 60 grados.

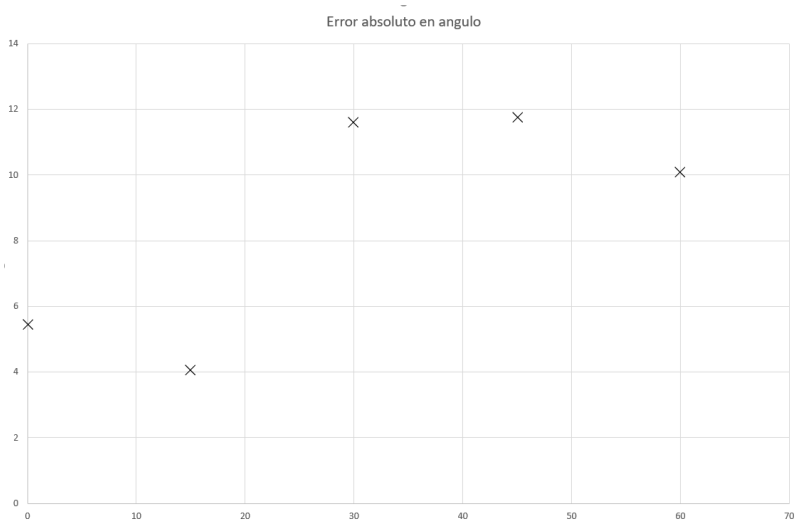

Figura 6.4 Gráfica de error en función del ángulo de observación.

6.2 Caso: 2 marcadores

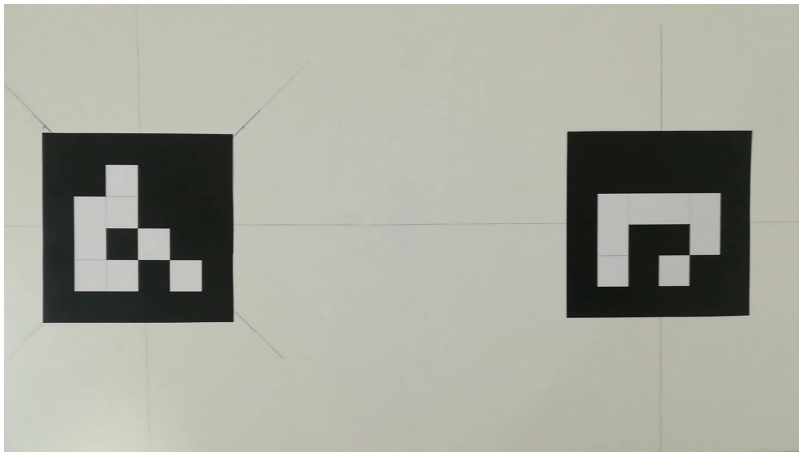

Figura 6.5 Prueba con 2 marcadores.

Ejecutando el programa de mapeado obtenemos una gráfica de dispersión con los resultados.

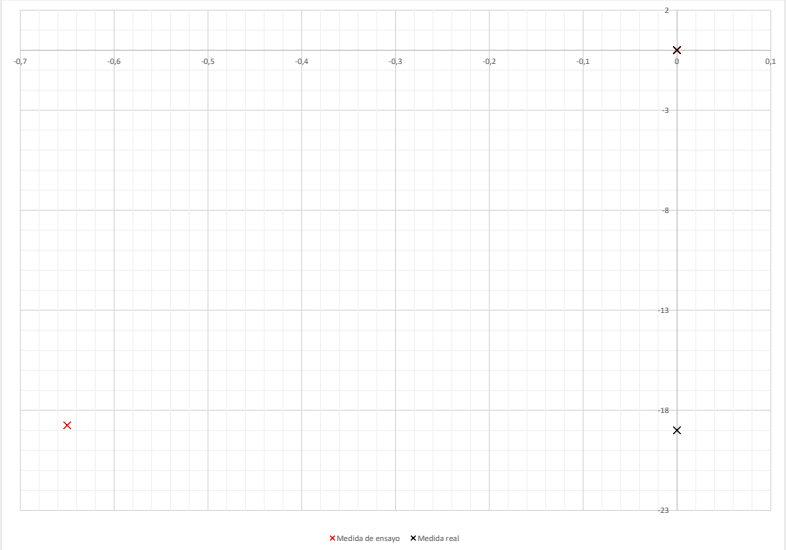

Figura 6.6 Resultados con 2 marcadores.

En este caso son dos marcadores alineados a una distancia de 19 cm por lo que la desviación de la medida es de unos 0.7 cm, esta desviación es el valor típico de error que se acumula en medidas relativas entre marcadores.

También aprovechamos esta ocasión para comprobar que el grafo generado es coherente como se puede observar en la representación gráfica de un grafo usando la herramienta matlab.

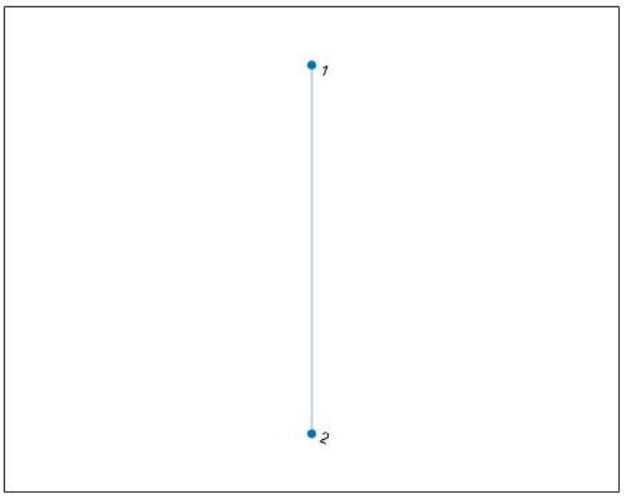

Figura 6.7 Grafo con 2 marcadores.

Ya que en estas condiciones solo poseemos dos marcadores solo se muestran dos nodos los cuales están conectados porque en todo momento de la grabación estos dos marcadores se veían mutuamente.

6.3 Caso: 3 marcadores

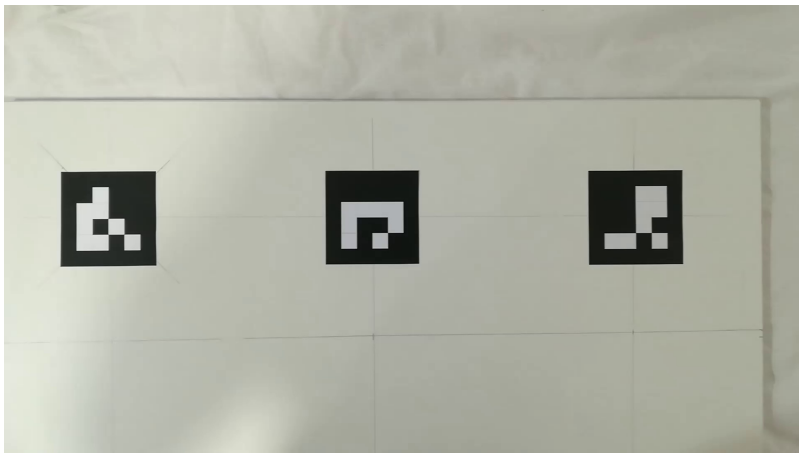

Figura 6.8 Prueba con 3 marcadores.

Son 3 marcadores alineados esta prueba se realiza para demostrar que el error de medición aumenta cuanto mas lejos estén entre sí los marcadores, exactamente es el caso anterior pero añadiendo un marcador mas. Por lo tanto obtenemos que para una distancia de unos 20 cm de distancia entre marcadores el error acumulado es de menos de 1 cm es muy buen resultado por no decir que acabamos de comprobar que el calculo de distancias relativas se hace de forma correcta.

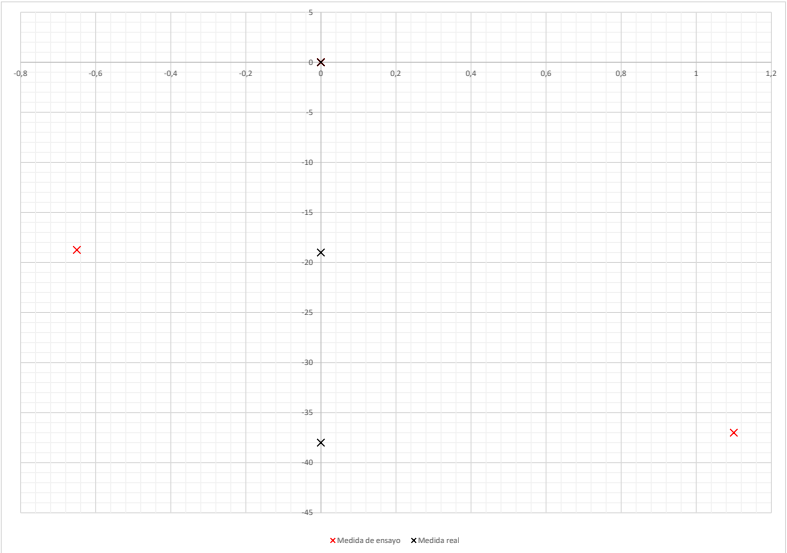

Figura 6.9 Resultados con 3 marcadores.

Como cabe de esperar los resultados son los mismos para los dos primeros marcadores pero para el tercero tenemos como resultado x=39 cm y=1.1 cm con esto y teniendo en cuenta que se encuentra a 38 cm obtenemos una desviación de 1.4 cm, al aumentar el doble la distancia obtenemos una desviación del doble por lo que podemos asumir que existe una relación lineal. Por lo que esto se puede extrapolar para mapeados similares una gran ventaja para estimar de previamente que error dispondrá un mapeado de ciertas dimensiones.

6.4 Caso: 6 marcadores

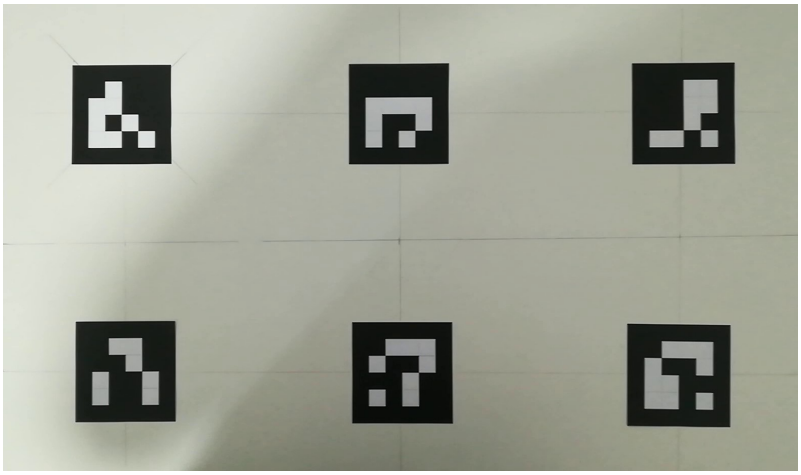

Figura 6.10 Prueba con 6 marcadores.

Con el objetivo de comprobar el comportamiento con mas marcadores se hace una prueba con 6 marcadores.

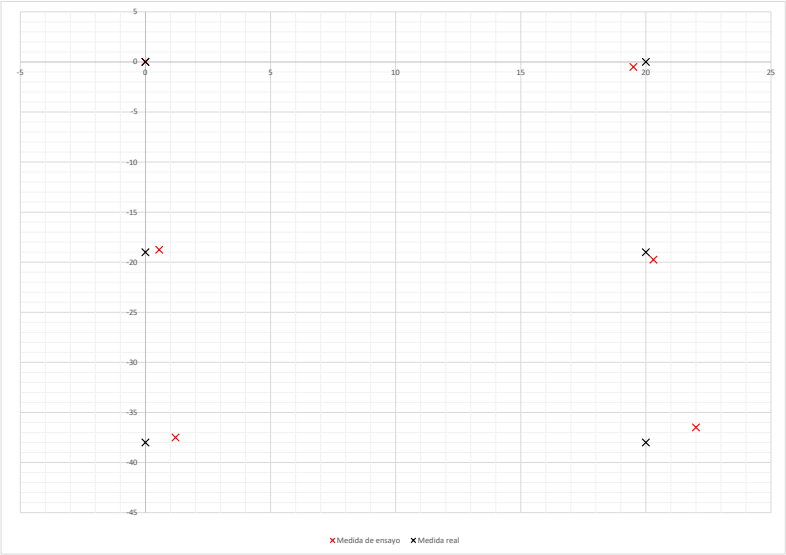

Figura 6.11 Resultados con 6 marcadores.

Como era esperar existen mas desviaciones que con solo 3 marcadores pero siguen dentro de lo esperado, puesto que el vídeo se ha realizado a mas distancia que con menos marcadores, la detección de los mismos acumulan mas error, pero este sigue siendo lineal, en el caso del marcador mas alejado acumula un error absoluto máximo de 1.6 cm. Puesto que este se encuentra en diagonal podemos seguir afirmando que el error acumulado es lineal y por tanto verificamos la hipótesis del apartado anterior para realizar estimaciones de error en mapeo de características similares.

6.5 Caso: Visión incompleta de marcadores

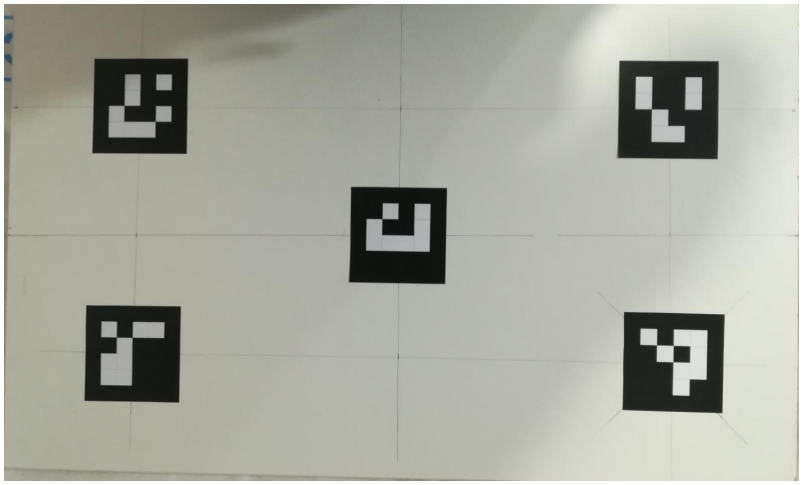

Figura 6.12 Prueba con 5 marcadores.

Para comprobar que el programa puede generar un mapeo adecuado en el que la localización de todos los marcadores en una misma imagen es imposible, obtenemos la posición de los marcadores de la derecha conociendo la distancia relativa al marcador central puesto que en ningún momento del vídeo se observan los marcadores de la derecha e izquierda al mismo tiempo.

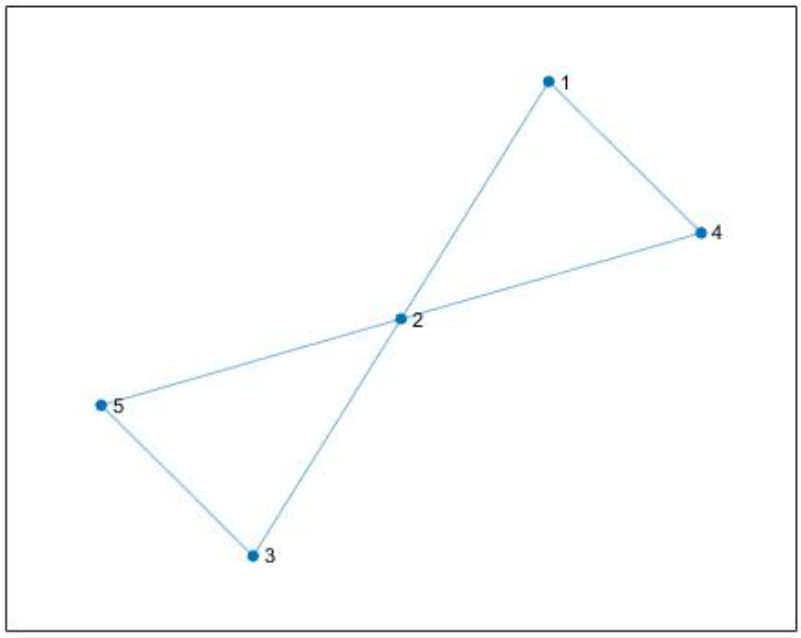

Figura 6.13 Grafo con 5 marcadores.

Para comprobar este comportamiento se extrae el grafo generado por el algoritmo confirmando que para llegar del nodo 1,2 al 4,5 debemos pasar por el nodo puente 3. Esta gráfica se a realizado usando el programa matlab pues es mas amigable en cuanto a graficar grafos se refiere.

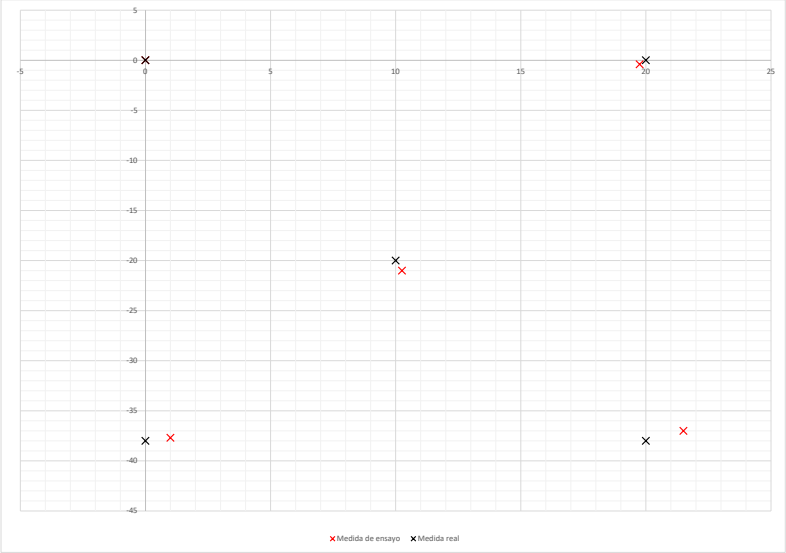

Figura 6.14 Resultado con 5 marcadores.

Los resultados son mejores que con 6 marcadores pero esto es debido que el vídeo es tomado mucho mas cerca de los marcadores por lo que la precisión en su detección aumenta considerablemente dando lugar a un error absoluto máximo de 1.5 cm. Y de nuevo volvemos a comprobar q esta relación es bastante lineal. Con este ensayo podemos cerciorarnos definitivamente que el algoritmo de mapeado no dispones de errores de concepto, aunque todavía queden aspectos que mejorar.

6.6 Caso: Detección en esquinas

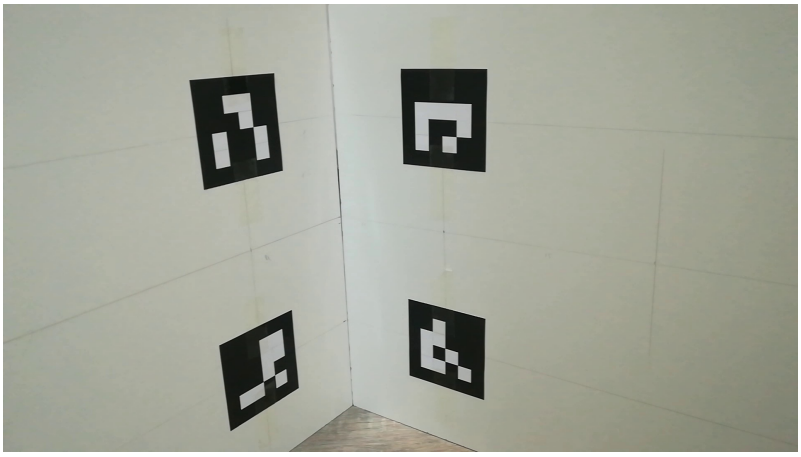

Figura 6.15 Prueba con 2 planos.

Colocamos 2 marcadores por lado de la esquina y se procede a grabar dicha disposición.

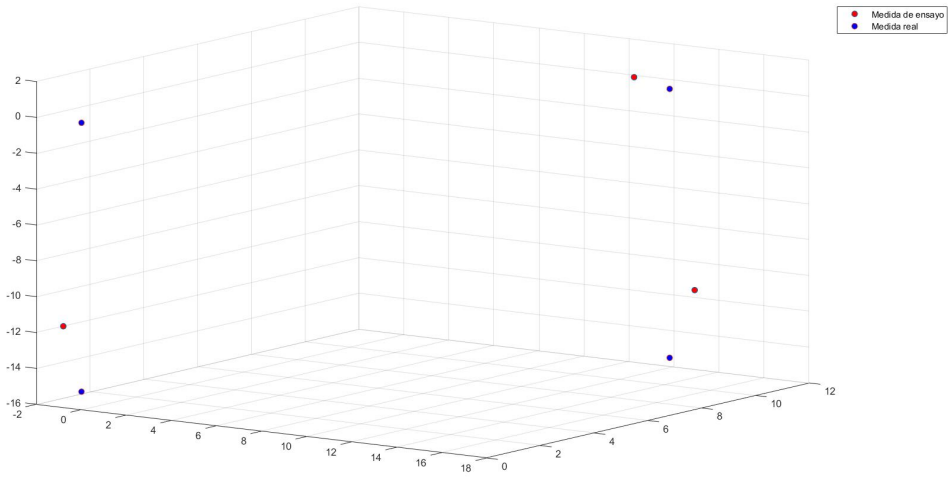

Figura 6.16 Resultado con 2 planos.

Los resultados son bastante mas pobres debido principalmente a la distancia de grabación a los marcadores y el ángulo del marcador con respecto a la cámara.

Aruco es un modulo que calcula muy bien la posición del marcador y la orientación sobre el plano, sin embargo cuando calcula el eje Z que es el perpendicular el error es bastante mayor,como ya comentamos aún teniendo en cuenta esto, los resultados no son malos, puesto que el mayor error se encuentra para el marcador mas alejado del de referencia el cual posee una desviación de unos 6 cm. NO ha podido implementar en este caso la mejora del eje Z puesto que en ninguna de las dos superficies posee 3 o mas marcadores.

Esto es un claro ejemplo del problema que se comentaba en la sección 6.1.2 puesto que el ángulo mínimo con el que se pueden tomar las imágenes son de 45º teniendo en cuenta esto el error acumulado es significativo sobre todo si en alguna momento de la grabación se supera ese ángulo límite de 60 º o simplemente te acercas a ese valor.

6.7 Caso a mayor escala

Para esta prueba se ha usado una cámara de mayor calidad hasta ahora se ha usado una cámara de 6 Mpx pero para esta prueba se ha usado una de 48 Mpx obteniendo una mejora de las medidas bastante sustancial.

Figura 6.17 Ensayo a mayor escala.

Superponiendo el resultado con los datos reales obtenemos la siguiente gráfica con la que podemos apreciar una precisión bastante buena acumulando un error absoluto máximo de 3.96 cm para el marcador mas alejado un resultado mas que razonable. Podemos sacar la conclusión que para mapeados en una sola superficie se puede extender bastante el área a mapear pudiendo hacerse uso de estos marcadores el los techos de las fabricas, puesto que es solo un plano permite hacer un mapeado preciso en mayor área.

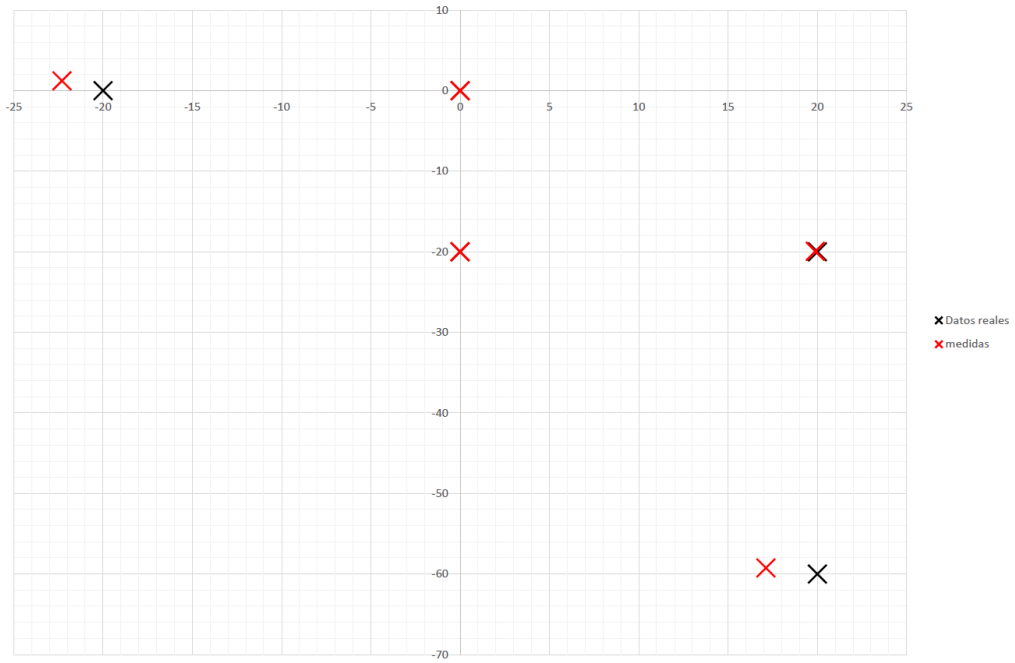

Figura 6.18 Resultado a mayor escala.

7 Conclusiones y lineas futuras

7.1 Conclusiones

Con estos dos módulos realizados disponemos tanto de una interfaz para seguir la posición de nuestro drone en interiores así como el software para realizar el mapeado en interiores. aunque el mapeado no es todo lo eficiente que desearíamos para una aplicación industrial, sirve para vislumbrar la posibilidad de hacer posicionamiento indoor con gran precisión a costa de que el mapeado sea lo mas preciso posible.

El posicionamiento logrado con este proyecto nos da la capacidad de crear pequeñas áreas mapeadas, las cuales se pueden usar para ubicar un drone con alta precisión. Una de las posibles aplicaciones de estos programas con las características actuales, serían realizar despegues y aterrizajes autónomos de gran precisión. Esta es una operación altamente compleja y que normalmente con lleva el mayor riesgo en las operaciones por lo que es bastante significativo.

Otro uso podría ser el refuerzo de otras tecnologias de posicionamiento como la anteriormente mencionada UWB. Si existen zonas a las que es complicado que alcancen este tipo de posicionamiento debido a que existan interferencias electromagnéticas en zonas concretas esta tecnología podria ser usada como sustituto.

Ya que en estos programas cabe mucho margen de mejora, sacando su máximo partido se podría llegar a su uso de forma industrial. Buscando empresas de servicios que usan drones dentro de fabricas encontramos un sector el cual se vería muy beneficiado de la aplicación de esta tecnología, este es el caso de la empresa ON Identity que se dedica al uso de drones con antenas RFID para realizar el inventario de almacenes. Para esta tarea usan drones pilotados por expertos en manejo de este tipo de aeronaves en interiores. Si estos drones dispusieran de la capacidad de posicionarse de forma precisa se podría realizar vuelos periódicos y totalmente autónomos con el objetivo de agilizar la logística de una factoria.

7.2 Lineas futuras

Como ya hemos comentado es un programa que dista de considerarse optimizado, de entre las formas claras de mejorar el programa existen dos.En primer lugar crear el concepto de calidad de una imagen puesto que no es lo mismo tomar una imagen desde muy cerca que de lejos pues los errores se pueden agrabar considerablemente, este factor de calidad dependería de una funcion que evaluara los par ametros de contraste. lejania al marcador y orientación con respecto a este para determinar cuan buena es la medida y así poder desechar aquellas que no entren dentro de un rango aceptable. Para ello se tendría que realizar un estudio detallado de los errores en funcion de los anteriores parametros. Al implementar esto reduciriamos significativamente el problema de acumular excesivos errores en el mapeo. Y la otra opción es modificar la librería misma de aruco modificando la forma en la que calculan el vector rotación porque al igual que los angulos de euler tiene problemas en cuanto a estabilidad numérica. Este punto modificaría significativamente el problema principal del algoritmo actual.

En cuanto a la interfaz gráfica sería conveniente añadir menus para realizar el mapeado y la carga de datos de nubes de puntos desde la aplicación y no modificando el programa. Para ello sería interesante usar librerías mas potentes en desarrollo de entornos graficos que la que actualmente se esta haciendo uso.

Así como la implementación de un sistema soporte a la visualización de marcadores puesto que es probable que en todo momento no pueda tener visión directa de marcadores ahí entra en juego la tecnología RFID si a cada marcador asocias una etiqueta RFID tendrás el mapeo de estas etiquetas automáticamente al hacer el mapeado de marcadores con la ventaja de que estos no necesitan visión directa.

Anexo: Codigo Python

En este anexo se encuentran los diferentes programas usados para el desarrollo de este proyecto.

Código 7.1 Modulo de calibración.

```
# criterio de calibracion
ERROR=1
m=chess_size[0]
n=chess_size[1]
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objp = np.zeros((m*n,3), np.float32)
objp[:,:2] = np.mgrid[0:m,0:n].T.reshape(-1,2)
# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob("chess*."+img_format)
n_img=len(images)
print("Número de imágenes de calibración encontradas: {}".format(n_img))
n_imagOK=0;
for fname in tqdm(images):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # encontrar esquinas
    ret, corners = cv2.findChessboardCorners(gray, (9,6),None)
    n_imagOK=n_imagOK+ret
    # Si encuentras añade las esquinas
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # dibujar esquinas
        img = cv2.drawChessboardCorners(img, (9,6), corners2,ret)

        if mostrar :
            cv2.imshow('img',img)
            cv2.waitKey(500)
        cv2.imwrite("corner.png",img)
print("\n")
print("Imágenes validas de calibración: {}/{}".format(n_imagOK,n_img))
```

```
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera ( objpoints , imgpoints
        , gray.shape [:: - 1 ], None, None)
    print("Calibración exitosa.")
    cv2 . destroyAllWindows ()
    '''
    print(mtx)
    print(dist)
    print(ret)
    '''
    return mtx, dist, ret
```

Código 7.2 Programa de calculo de error en distancias.

```
import os
import pandas as pd
import numpy as np
import glob
import cv2
import cv2.aruco as aruco
from  tqdm import tqdm
import time
import modulo
import math
import sys
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import scipy.io as sio

mtx, dist, ret =  modulo.calibrar("jpg",[9,6],0)
#save_coefficients(mtx, dist, path)
#mtx,dist=load_coefficients(path)
aruco_dict     =  aruco.Dictionary_get(aruco.DICT_4X4_50)
arucoParameters = aruco.DetectorParameters_create()


size_marker = 18
frames     = glob.glob("frame*.jpg")
n_frames   = len(frames)

for frame in tqdm(frames):
    ids    = [[0]]
    imagen = cv2.imread(frame)

    frame_gray                          = cv2.cvtColor(imagen, cv2.COLOR_BGR2
        GRAY)
    marker_corners, ids, rejectedImgPoints = aruco.detectMarkers(frame_gray,
        aruco_dict, parameters=arucoParameters)
    rvec, tvec, obj_points              = aruco.estimatePoseSingleMarkers(
        marker_corners,size_marker, mtx, dist)
    print(tvec)
```

Código 7.3 Programa de calculo de error en angulos.

```
import os
import pandas as pd
```

```
import numpy as np
import glob
import cv2
import cv2.aruco as aruco
from  tqdm import tqdm
import time
import modulo
import math
import sys
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import scipy.io as sio

mtx, dist, ret =  modulo.calibrar("jpg",[9,6],0)
#save_coefficients(mtx, dist, path)
#mtx,dist=load_coefficients(path)
aruco_dict     =  aruco.Dictionary_get(aruco.DICT_4X4_50)
arucoParameters = aruco.DetectorParameters_create()


size_marker = 18
frames     = glob.glob("IMG*.jpg")
n_frames   = len(frames)

for frame in tqdm(frames):
    ids    = [[0]]
    imagen = cv2.imread(frame)

    frame_gray                          = cv2.cvtColor(imagen, cv2.COLOR_BGR2
        GRAY)
    marker_corners, ids, rejectedImgPoints = aruco.detectMarkers(frame_gray,
        aruco_dict, parameters=arucoParameters)
    rvec, tvec, obj_points              = aruco.estimatePoseSingleMarkers(
        marker_corners,size_marker, mtx, dist)
    R,_=cv2.Rodrigues(rvec)
    axZ=R[:,[2]]
    print(180-math.acos(axZ[2]/(axZ[0]**2+axZ[1]**2+axZ[2]**2))*57.29)
```

Código 7.4 Programa de mapeo.

```
import os
import pandas as pd
import numpy as np
import glob
import cv2
import cv2.aruco as aruco
from  tqdm import tqdm
import time
import modulo
import math
import sys
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import scipy.io as sio
from CalibejeZ import*
```

```
def save_coefficients(mtx, dist, path):
    """ Save the camera matrix and the distortion coefficients to given path/
        file. """
    cv_file = cv2.FileStorage(path, cv2.FILE_STORAGE_WRITE)
    cv_file.write("K", mtx)
    cv_file.write("D", dist)
    # note you *release* you don't close() a FileStorage object
    cv_file.release()
def load_coefficients(path):
    """ Loads camera matrix and distortion coefficients. """
    # FILE_STORAGE_READ
    cv_file = cv2.FileStorage(path, cv2.FILE_STORAGE_READ)

    # note we also have to specify the type to retrieve other wise we only get
        a
    # FileNode object back instead of a matrix
    camera_matrix = cv_file.getNode("K").mat()
    dist_matrix = cv_file.getNode("D").mat()

    cv_file.release()
    return [camera_matrix, dist_matrix]

mtx, dist, ret =  modulo.calibrar("jpg",[9,6],0)
path='F:/Escritorio_pen/TFG/Scripts/Mapeado/Eje Z'
#save_coefficients(mtx, dist, path)
#mtx,dist=load_coefficients(path)
aruco_dict     =  aruco.Dictionary_get(aruco.DICT_4X4_50)
arucoParameters = aruco.DetectorParameters_create()
#RUTA DONDE SE ALMACENAN LOS FRAMES
os.chdir(r'F:/Escritorio_pen/TFG/Scripts/Mapeado/Eje Z/frames')


size_marker = 7
frames     = glob.glob("*.jpg")
n_frames   = len(frames)
df         = pd.DataFrame(columns=('pareja', 'pos1', 'gir1', 'pos2', 'gir2'))


for frame in tqdm(frames):
    ids    = [[0]]
    imagen = cv2.imread(frame)

    frame_gray                          = cv2.cvtColor(imagen, cv2.COLOR_BGR2
        GRAY)
    marker_corners, ids, rejectedImgPoints = aruco.detectMarkers(frame_gray,
        aruco_dict, parameters=arucoParameters)
    rvec, tvec, obj_points              = aruco.estimatePoseSingleMarkers(
        marker_corners,size_marker, mtx, dist)


    if  len(marker_corners)>0:

        n_tags = len(ids)
        if n_tags>1:
            #rvec,tvec = calib_ejeZ(rvec,tvec,ids,40)

            for i in range(n_tags-1):
```

```
            for j in range(i+1,n_tags):
                rvec1 = rvec[i,0,:]
                rvec2 = rvec[j,0,:]
                R1,_=cv2.Rodrigues(rvec1)
                R2,_=cv2.Rodrigues(rvec2)
                az1= R1[:,[2]]
                az2= R2[:,[2]]

                if ids[i,0]<ids[j,0]:
                    df.loc[len(df)] = [ids[i,0]*100+ids[j,0],tvec[i,0,:],rvec
                        1,tvec[j,0,:],rvec2]
                    #df={'pareja': ids[i,0]*100+ids[j,0],'pos1':rvec[i,0,:],'
                        gir1': tvec[i,0,:],'pos2':rvec[j,0,:],'gir2': tvec[j
                        ,0,:]}
                    #df.append(df,ignore_index=True)
                else:
                    df.loc[len(df)] = [ids[j,0]*100+ids[i,0],tvec[j,0,:],rvec
                        2,tvec[i,0,:],rvec1]

df.sort_values('pareja', inplace=True)


#OBTENEMOS DISTACIAS ENTRE MARCADORES PARA ELLO MODIFICAMOS EL DATAFRAME

#definimos la funcion q calcula la posicion entre tags



def relativePosition(x):
    rvec1 = x[2]
    tvec1 = x[1]
    rvec2 = x[4]
    tvec2 = x[3]

    d   = tvec2-tvec1
    R1,_ = cv2.Rodrigues(rvec1)
    R2,_ = cv2.Rodrigues(rvec2)
    dd  = np.dot(R1,d)
    v1=R1[:,2]
    v2=R2[:,2]
    '''
    if angle(v1, v2)<3.14*0.5:
        dd[2]=0
    '''
    return dd

def R12(x):
    rvec1 = x[2]
    rvec2 = x[4]
    Rf1,_=cv2.Rodrigues(rvec1)
    Rf2,_=cv2.Rodrigues(rvec2)
    R12=np.dot(np.transpose(Rf1),Rf2)
    R12vec,_=cv2.Rodrigues(R12)
    return R12vec[:,0]

def R21(x):
```

```
    rvec1 = x[2]
    rvec2 = x[4]
    Rf1,_=cv2.Rodrigues(rvec1)
    Rf2,_=cv2.Rodrigues(rvec2)
    R21=np.transpose(np.dot(np.transpose(Rf1),Rf2))
    R21vec,_=cv2.Rodrigues(R21)
    return R21vec[:,0]


df_pos                = df.apply(relativePosition, axis = 1)
df_pos                = df_pos.to_frame(name='P')
df_pos[['x','y','z']] = pd.DataFrame(df_pos.P.tolist(), index=df_pos.index)
df_pos                = pd.concat([df.pareja,df_pos[['x','y','z']]],axis=1)
df_pos[['gx12','gy12','gz12']] = pd.DataFrame(df.apply(R12, axis = 1).tolist(),
    index=df_pos.index)
df_pos[['gx21','gy21','gz21']] = pd.DataFrame(df.apply(R21, axis = 1).tolist(),
    index=df_pos.index)
#df[['x','y','z']] = pd.DataFrame(df_pos[0].tolist(), index=df.index)

df_mean = pd.DataFrame()
df_mean['mx'] = df_pos.groupby('pareja')['x'].mean()
df_mean['my'] = df_pos.groupby('pareja')['y'].mean()
df_mean['mz'] = df_pos.groupby('pareja')['z'].mean()
df_mean['dx'] = df_pos.groupby('pareja')['x'].std()
df_mean['dy'] = df_pos.groupby('pareja')['y'].std()
df_mean['dz'] = df_pos.groupby('pareja')['z'].std()

df_mean['mgx12'] = df_pos.groupby('pareja')['gx12'].mean()
df_mean['mgy12'] = df_pos.groupby('pareja')['gy12'].mean()
df_mean['mgz12'] = df_pos.groupby('pareja')['gz12'].mean()
df_mean['dgx12'] = df_pos.groupby('pareja')['gx12'].std()
df_mean['dgy12'] = df_pos.groupby('pareja')['gy12'].std()
df_mean['dgz12'] = df_pos.groupby('pareja')['gz12'].std()

df_mean['mgx21'] = df_pos.groupby('pareja')['gx21'].mean()
df_mean['mgy21'] = df_pos.groupby('pareja')['gy21'].mean()
df_mean['mgz21'] = df_pos.groupby('pareja')['gz21'].mean()
df_mean['dgx21'] = df_pos.groupby('pareja')['gx21'].std()
df_mean['dgy21'] = df_pos.groupby('pareja')['gy21'].std()
df_mean['dgz21'] = df_pos.groupby('pareja')['gz21'].std()

#Ahora filtramos los valores que salgan de la media 2 veces desviación típica

def eliminar(x):
  index=x.pareja
  margen=1.5

  mx  = df_mean['mx'][index]
  my  = df_mean['my'][index]
  mz  = df_mean['mz'][index]

  dx  = df_mean['dx'][index]*margen
  dy  = df_mean['dy'][index]*margen
  dz  = df_mean['dz'][index]*margen

  e_x = abs(x.x-mx)
  e_y = abs(x.y-my)
```

```
  e_z = abs(x.z-mz)
  if e_x>dx or e_y>dy or e_z>dz:
    return 1
  return 0

df_pos['delete'] = df_pos.apply(eliminar, axis = 1)
index=df_pos['delete'] == 1
df_pos.drop(df_pos[index].index,axis=0)

df_mean = pd.DataFrame()
df_mean['mx']=df_pos.groupby('pareja')['x'].mean()
df_mean['my']=df_pos.groupby('pareja')['y'].mean()
df_mean['mz']=df_pos.groupby('pareja')['z'].mean()
df_mean['dx']=df_pos.groupby('pareja')['x'].std()
df_mean['dy']=df_pos.groupby('pareja')['y'].std()
df_mean['dz']=df_pos.groupby('pareja')['z'].std()

df_mean['mgx12'] = df_pos.groupby('pareja')['gx12'].mean()
df_mean['mgy12'] = df_pos.groupby('pareja')['gy12'].mean()
df_mean['mgz12'] = df_pos.groupby('pareja')['gz12'].mean()
df_mean['dgx12'] = df_pos.groupby('pareja')['gx12'].std()
df_mean['dgy12'] = df_pos.groupby('pareja')['gy12'].std()
df_mean['dgz12'] = df_pos.groupby('pareja')['gz12'].std()

df_mean['mgx21'] = df_pos.groupby('pareja')['gx21'].mean()
df_mean['mgy21'] = df_pos.groupby('pareja')['gy21'].mean()
df_mean['mgz21'] = df_pos.groupby('pareja')['gz21'].mean()
df_mean['dgx21'] = df_pos.groupby('pareja')['gx21'].std()
df_mean['dgy21'] = df_pos.groupby('pareja')['gy21'].std()
df_mean['dgz21'] = df_pos.groupby('pareja')['gz21'].std()

df_mean['m1']=df_mean.index//100
df_mean['m2']=df_mean.index-(df_mean.index//100)*100


# Definimos el grafo
os.chdir(r'F:/Escritorio_pen/TFG/Scripts/Mapeado')
os.system('clear')


from graph.Graph import Graph
from algorithms_prev.dijkstra_prev import *
from algorithms_prev.floyd_warshall_prev import *


sources=df_mean['m1'].tolist()+df_mean['m2'].tolist()
targets=df_mean['m2'].tolist()+df_mean['m1'].tolist()
weights=(df_mean['mx']**2+df_mean['my']**2+df_mean['mz']**2).tolist()
sio.savemat('grapho.mat', {'s': sources, 't': targets})
for i in range(len(weights)):
    weights[i]=math.sqrt(weights[i])
weights=weights+weights

graph = Graph(sources, targets, weights)

print("----------------------------------------")
```

```
graph.print_r()
print("----------------------------------------")
dist, prev2 = Floyd_Warshall_PREV(graph)
print(prev2)


#Creada la matriz de predecesores creamos los caminos en un principio no debe
    saler ningun -1

pA=prev2[0,:]
nNodos=len(pA)
CAM=np.ones((nNodos-1, nNodos))*-1

for iA in range(1,len(pA)): #Recorremos todo el vector menos el primer elemento
    j=0
    CAM[iA-1,j]=iA
    nodo=iA
    while nodo != 0:
        j=j+1
        nodo=pA[nodo]
        CAM[iA-1,j]=nodo


#Ya tenemos los caminos creados pasamos a las posiciones
def create_grafox(col):
    x=np.array([[col.mx],[col.my],[col.mz]])
    return x

def create_grafoR12(col):
    r12,_=cv2.Rodrigues(np.array([col.mgx12,col.mgy12,col.mgz12]))
    return r12

def create_grafoR21(col):
    r12,_=cv2.Rodrigues(np.array([col.mgx12,col.mgy12,col.mgz12]))
    r21=np.transpose(r12)
    return r21


df_grafo=pd.DataFrame(columns=('X','R12','R21'))
df_grafo['X']=df_mean.apply(create_grafox, axis = 1)
df_grafo['R12']=df_mean.apply(create_grafoR12, axis = 1)
df_grafo['R21']=df_mean.apply(create_grafoR21, axis = 1)
DIST_G=pd.DataFrame(columns=('X','R'))
DIST_G.loc[0]=[np.array([[0],[0],[0]]),np.array([[1,0,0],[0,1,0],[0,0,1]])]

#Algoritmo
def calc_pos(M2,M1):
    if M2<M1:
        pareja=M2*100+M1
        RM='R21'
    else:
        pareja=M1*100+M2
        RM='R12'

    R=np.dot(df_grafo[RM][pareja],DIST_G['R'][M1])
    sol=[DIST_G['X'][M1]+np.dot(DIST_G['R'][M1],df_grafo['X'][pareja]),R]
    return sol
```

```
for  i in range(0,nNodos-1):
    j=nNodos-1
    n1= -1
    while n1== -1:
        n1 =CAM[i,j]
        j=j-1

    for jj in range(j,-1,-1):
        M2=CAM[i,jj]
        M1=CAM[i,jj+1]
        aux=calc_pos(M2,M1)
        DIST_G.loc[M2]=[aux[0].tolist(),aux[1].tolist()]

DIST_G
print(DIST_G)
#Ya tenemos toda la info ordenada ahora empezamos el algoritmo de mapeado

def sepx(y):
  return np.array(y.X[0][0])

def sepy(y):
  return np.array(y.X[1][0])

def sepz(y):
  return np.array(y.X[2][0])

x=DIST_G.apply(sepx, axis = 1).tolist()
y=DIST_G.apply(sepy, axis = 1).tolist()
z=DIST_G.apply(sepz, axis = 1).tolist()


fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')
ax1.scatter(x, y, z, c='g', marker='o')
plt.show()
```

Código 7.5 Funciones para la optimización del calculo del eje Z.

```
import numpy as np
import pandas as pd
import math
import cv2
#grados se refiere al angulo entre planos que se consideran mismo plano
#Comprobar si rvec y tvec son listas o arrays suponemos q son arrays
def calib_ejeZ(rvec,tvec,ids,grados):
  n_tags = len(ids)
  Planos=pd.DataFrame(columns=('Plano', 'markers'))
  n_plano=0
  list_used=[0]*n_tags
  for i in range(n_tags):
    list_marker=[]
    if list_used[i]== 0:
      list_marker.append(i)
      Planos.loc[len(Planos)] = [n_plano,list_marker]
      list_used[i]=1
```

```
      n_plano=n_plano+1
      for j in range(i+1,n_tags):
        if list_used[j]== 0:
          rvec1 = rvec[i,0,:]
          rvec2 = rvec[j,0,:]
          if same_plane(rvec1,rvec2,grados)==1:
            Planos.markers.iloc[-1].append(j)
            list_used[j]=1


  #He creado un dataframe en el q cada fila tiene un plano y en la segunda una
  #columna una lista con los marcadores de ese plano
    for i in range(0,n_plano):
        plane_markers=Planos['markers'][i]
        rvec,tvec=opt_matrix(rvec,tvec,plane_markers)

  return rvec,tvec


#funciones auxiliares
def dotproduct(v1, v2):
    return sum((a*b) for a, b in zip(v1, v2))

def length(v):
    return math.sqrt(dotproduct(v, v))

def angle(v1, v2):
    return math.acos(dotproduct(v1, v2) / (length(v1) * length(v2)))
    if (a==a).any():
        return 0
    else:
        return math.acos(dotproduct(v1, v2) / (length(v1) * length(v2)))
def same_plane(rvec1,rvec2,grados):
  R1,_ = cv2.Rodrigues(rvec1)
  R2,_ = cv2.Rodrigues(rvec2)
  v1=R1[:,2]
  v2=R2[:,2]
  if angle(v1,v2)<grados*(3.14/180):
    return 1
  else:
    return 0


def opt_matrix(rvec,tvec,list_marker):
  n_points= len(list_marker)
  print(list_marker)
  if n_points >2:
    points=np.reshape(tvec[list_marker ,:],(n_points,3))
    A=np.ones((n_points,3))
    A[:,[0,1,2]]=points[:,[0,1,2]]
    b=np.ones((n_points,1))*-1
    if n_points>3:
      condA=np.linalg.cond(np.dot(np.transpose(A),A))
      if condA<500:
        x=np.linalg.lstsq(A,b,rcond= None)[0] # Nos proporciona los
            coeficientes del plano Z=Ax +By +C
      else:
```

```
      return rvec,tvec
  else:
    condA=np.linalg.cond(A)
    if condA<500:
      x=np.linalg.solve(A,b)
    else:
      return rvec,tvec
  axZ=np.array([[x[0,0]],[x[1,0]],[x[2,0]]])
  axZ=axZ/math.sqrt(x[0,0]**2+x[1,0]**2+x[2,0]**2)
  for i  in range(n_points):
    R,_=cv2.Rodrigues(rvec[list_marker[i],0,:])
    axX=R[:,[0]]
    axY=R[:,[1]]
    R[:,[2]]=axZ

    axX=axX-np.vdot(axX,axZ)*axZ
    axY=axY-np.vdot(axY,axZ)*axZ

    R[:,[0]]=axX
    R[:,[1]]=axY
    R=np.array(R)
    trvec,_=cv2.Rodrigues(R)
    rvec[list_marker[i],0,:]=np.reshape(trvec,(1,3))
    #Una vez fijado el plano ajustamos tvec para q pertenezcan a dixo plano
    landa=(-1-(x[0,0]*tvec[i,0,0]+x[1,0]*tvec[i,0,1]+x[2,0]*tvec[i,0,2]))/(x
        [0,0]**2+x[1,0]**2+x[2,0]**2)
    tvec[i,0,:]=tvec[i,0,:]+landa*np.array([x[0,0],x[1,0],x[2,0]])
    sol=tvec[i,0,0]*x[0,0]+tvec[i,0,1]*x[1,0]+tvec[i,0,2]*x[2,0]+1
return rvec,tvec
```

Código 7.6 Programa principal para grafos.

```
#Estos codigos han sido extraidos de la pagina web: http://micaminomaster.com.
    co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/

from graph.DynamicIncrementalGraph import DynamicIncrementalGraph
import numpy as np


class Graph(DynamicIncrementalGraph):

    def __init__(
        self,
        source=[],
        target=[],
        weight=[],
        directed=True,
        set_nodes_with_num_nodes=0
    ):
        DynamicIncrementalGraph.__init__(self, source, target, weight, directed,
             set_nodes_with_num_nodes)

    def print_r(self):
        print("Source: ", self.source)
        print("Target: ", self.target)
        print("Weight: ", self.weight)
        print("Vertex: ", self.nodes)

    def stats(self):
        edges = len(self.source)
        nodes = len(self.nodes)
        print("Total Nodes: ", nodes)
        print("Total Edges: ", edges)
        print("Density:", edges / (nodes*nodes))

    def get_density(self):
        edges = len(self.source)
        nodes = len(self.nodes)
        return edges / (nodes*nodes)

    def export(self):
        return [(int(self.source[i]), int(self.target[i]), self.weight[i]) for
            i in range(self.source.size)]

    def export_values(self):
        directed = 'true' if self.directed else 'false'

        return {
            'source': list(np.array([str(int(x)) for x in self.source])),
            'target': list(np.array([str(int(x)) for x in self.target])),
            'weight': list(np.array([str(int(x)) for x in self.weight])),
            'nodes': list(np.array([str(int(x)) for x in self.nodes])),
            'density': self.get_density(),
            'directed': directed,
            'last_edge_action': self.last_edge_action,
            'last_edge_updated': list(np.array([str(int(x)) for x in self.last_
                edge_updated])),
```

```
            'last_node_action': self.last_node_action,
            'last_node_updated': {
                'node': "None" if self.last_node_updated['node'] is None else
                    self.last_node_updated['node'],
                'source': list(np.array([str(int(x)) for x in self.last_node_
                    updated['source']])),
                'target': list(np.array([str(int(x)) for x in self.last_node_
                    updated['target']]))

            }
        }

    @staticmethod
    def import_values(values):
        directed = True if values['directed'] == 'true' else False

        new_values = {
            'source': np.array([int(x) for x in values['source']]),
            'target': np.array([int(x) for x in values['target']]),
            'weight': np.array([int(x) for x in values['weight']]),
            'nodes': np.array([int(x) for x in values['nodes']]),
            'directed': directed,
            'last_edge_action': values['last_edge_action'],
            'last_edge_updated': np.array([int(x) for x in values['last_edge_
                updated']]),
            'last_node_action': values['last_node_action'],
            'last_node_updated': {
                'node': values['last_node_updated']['node'],
                'source': np.array([int(x) for x in values['last_node_updated']['
                    source']]),
                'target': np.array([int(x) for x in values['last_node_updated']['
                    target']])
            }
        }

        g = Graph(new_values['source'], new_values['target'], new_values['
            weight'], new_values['directed'])
        g.last_edge_action = new_values['last_edge_action']
        g.last_edge_updated = new_values['last_edge_updated']
        g.last_node_action = new_values['last_node_action']
        g.last_node_updated = new_values['last_node_updated']

        return g


    @staticmethod
    def creategraph(total_nodes, pro_edges, weights=[1, 2, 3, 4, 5, 6, 7, 8,
        9], directed=True):

        source = []
        target = []
        weight = []

        for i in range(total_nodes):
            for k in range(total_nodes):
                if k == i:
                    continue
```

```
            p = 1 - pro_edges
            has_edge = np.random.choice(2, 1, p=[p, pro_edges])[0]

            if not has_edge:
                continue

            probabilities = np.zeros(len(weights))
            probabilities = probabilities + (1 / len(weights))
            w = np.random.choice(weights, 1, p=probabilities)[0]

            source.append(i)
            target.append(k)
            weight.append(w)

    return Graph(source, target, weight, directed, set_nodes_with_num_nodes
        =total_nodes)

"""
Comprueba si la matriz de distancia y la de predecesores, coinciden
"""
@staticmethod
def testDistPred(dist, pred):

    for node_source in range(len(dist)):
        for node_target in range(len(dist)):
            if node_source == node_target:
                continue

            pred_list = pred[node_source]
            distance_must = dist[node_source, node_target]
            target = node_target

            if distance_must == np.inf:
                if pred_list[target] == -1:
                    continue
                else:
                    print(f'({node_source}, {node_target}) must be -1')
                    return False

            distance = 0
            while True:
                predecesor = pred_list[target]
                source = predecesor

                if predecesor != -1:
                    distance += dist[predecesor, target]

                if source == -1:
                    if distance_must != distance:
                        print(f'({node_source}, {node_target}) not distance
                            coincidence')
                        return False
                    break
                target = source

    return True
```

Código 7.7 Funcion complementaria para grafos 1.

```
#Estos codigos han sido extraidos de la pagina web: http://micaminomaster.com.
    co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/
import numpy as np
import math
from collections import deque

from graph.GraphPro import GraphPro


class DynamicGraph(GraphPro):
    last_edge_updated = np.array([])
    last_edge_action = ""
    last_node_updated = {
        'node': None,
        'source': np.array([]),
        'target': np.array([]),
    }
    last_node_action = ""

    def __init__(
        self,
        source=[],
        target=[],
        weight=[],
        directed=True,
        set_nodes_with_num_nodes=0
    ):
        GraphPro.__init__(self, source, target, weight, directed, set_nodes_
            with_num_nodes)
        self.clean_vars()

    def clean_vars(self):
        self.last_edge_updated = np.array([])
        self.last_edge_action = ""
        self.last_node_updated = {
            'node': None,
            'source': np.array([]),
            'target': np.array([]),
        }
        self.last_node_action = ""

    def dynamic_decreasing_random_edge(self):
        count_max = 100
        flag = 0
        while True:
            source = np.random.choice(self.nodes, 1)[0]
            choisen = self.target[source == self.source]
            if choisen.size != 0:
                target = np.random.choice(choisen, 1)[0]
                break
            flag = flag + 1
```

```
            if flag >= count_max:
                return -2

        return self.dynamic_decreasing_edge(source, target)

    def dynamic_decreasing_edge(self, source, target):
        self.clean_vars()

        index = np.where(np.logical_and(self.source == source, self.target ==
            target))[0][0]

        self.source = np.delete(self.source, index)
        self.target = np.delete(self.target, index)
        self.weight = np.delete(self.weight, index)

        self.last_edge_updated = np.array([source, target])
        self.last_edge_action = "DELETE"

        self.sort_sources()

        return {
            "last_edge_updated": self.last_edge_updated,
            "last_edge_action": self.last_edge_action
        }

    def dynamic_incremental_random_edge(self, weights=[1, 2, 3, 4, 5, 6, 7, 8,
        9]):

        count_max = 100
        flag = 0
        while True:
            source = np.random.choice(self.nodes, 1)[0]
            index_for_target = np.invert(np.logical_or(np.in1d(self.nodes, self.
                target[source == self.source]), self.nodes == source))

            choisen = self.nodes[index_for_target]
            if choisen.size != 0:
                target = np.random.choice(choisen, 1)[0]
                break
            flag = flag + 1
            if flag >= count_max:
                return -2

        w = np.random.choice(weights)
        return self.dynamic_incremental_edge(source, target, w)

    def dynamic_incremental_edge_middle(self, weight=1):
        self.clean_vars()

        mid_node = len(self.nodes) // 2

        return self.dynamic_incremental_edge(mid_node-1, mid_node, weight)

    def dynamic_incremental_edge(self, source, target, weight=1):
        self.clean_vars()

        self.source = np.append(self.source, source)
```

```
        self.target = np.append(self.target, target)
        self.weight = np.append(self.weight, weight)

        self.last_edge_updated = np.array([source, target, weight])
        self.last_edge_action = "ADD"

        self.sort_sources()

        return {
            "last_edge_updated": self.last_edge_updated,
            "last_edge_action": self.last_edge_action
        }

    def dynamic_incremental_random_node(self, num_edges=1, weights=[1, 2, 3, 4,
         5, 6, 7, 8, 9]):
        self.clean_vars()
        node = np.amax(self.nodes) + 1

        number = math.ceil(num_edges/2)
        sources = np.random.choice(self.nodes, number, replace=False)
        w_sources = np.random.choice(weights, number)

        number = num_edges - number
        if number <= 0:
            targets = []
            w_targets = []
        else:
            targets = np.random.choice(self.nodes, number, replace=False)
            w_targets = np.random.choice(weights, number)

        self.sort_sources()

        return self.dynamic_incremental_node(node, sources, w_sources, targets,
             w_targets)

    def dynamic_incremental_node(self, node, sources, w_sources, targets, w_
        targets):
        self.clean_vars()
        if self.nodes[self.nodes == node].size > 0:
            return -1

        sources = np.array(sources)
        targets = np.array(targets)
        self.source = np.concatenate((self.source, sources, np.full(targets.
            size, node)))
        self.target = np.concatenate((self.target, np.full(sources.size, node),
             targets))
        self.weight = np.concatenate((self.weight, w_sources, w_targets))
        self.nodes = np.append(self.nodes, node)

        self.last_node_updated['node'] = node
        self.last_node_updated['source'] = sources
        self.last_node_updated['target'] = targets
        self.last_node_action = "ADD"

        self.sort_sources()
```

```
        return self.last_node_updated

    def dynamic_decreasing_random_node(self):
        node = np.random.choice(self.nodes, 1)[0]
        return self.dynamic_decreasing_node(node)

    def dynamic_decreasing_node(self, node):
        self.clean_vars()
        if self.nodes[self.nodes == node].size == 0:
            return -1

        self.last_node_updated['node'] = node
        while True:
            indexes = np.where(np.logical_or(self.source == node, self.target ==
                 node))[0]
            if len(indexes) == 0:
                break
            source = self.source[indexes[0]]
            target = self.target[indexes[0]]
            self.source = np.delete(self.source, indexes[0])
            self.target = np.delete(self.target, indexes[0])

            self.last_node_updated['source'] = np.append(self.last_node_updated
                ['source'], source)
            self.last_node_updated['target'] = np.append(self.last_node_updated
                ['target'], target)

        self.last_node_action = "DELETE"

        self.sort_sources()

        return {
            "last_node_updated": self.last_node_updated,
            "last_node_action": self.last_node_action
        }

    def edge_update(self, source, target, weight=1):
        self.clean_vars()

        if len(self.weight) != 0:
            self.weight[np.logical_and(self.source == source, self.target ==
                target)] = weight
            if not self.directed:
                self.weight[np.logical_and(self.source == target, self.target ==
                     source)] = weight

        self.last_edge_updated = np.array([source, target, weight])
        self.last_edge_action = "UPDATE"

        self.sort_sources()

        return {
            "last_edge_updated": self.last_edge_updated,
            "last_edge_action": self.last_edge_action
        }

    def edge_update_random(self, weight=1):
```

```
        count_max = 100
        flag = 0
        while True:
            source = np.random.choice(self.nodes, 1)[0]
            index_for_target = np.logical_or(np.in1d(self.nodes, self.target[
                source == self.source]), self.nodes == source)
            choisen = self.nodes[index_for_target]

            if choisen.size != 0:
                target = np.random.choice(choisen, 1)[0]
                if self.get_weight(source, target) > 1:
                    break

            flag = flag + 1
            if flag >= count_max:
                return -2

        return self.edge_update(source, target, weight=weight)

    def find_source_target_worst_scenary_incremental_edge(self):
        node_sources = {}
        node_targets = {}
        for node in self.nodes:
            node_sources[node] = self.find_total(node, "sources")
            node_targets[node] = self.find_total(node, "targets")

        max_path_long = 0
        node_source = 0
        node_target = 0
        for x in self.nodes:
            for y in self.nodes:
                if x == y:
                    continue

                if self.get_weight(x, y) != np.inf:
                    continue

                total_path_long = node_sources[x] * node_targets[y]
                if total_path_long > max_path_long:
                    max_path_long = total_path_long
                    node_source = x
                    node_target = y

        return {
            "source": node_source,
            "target": node_target
        }

    def find_edge_worst_scenary_update_edge(self):
        node_sources = {}
        node_targets = {}
        for node in self.nodes:
            node_sources[node] = self.find_total(node, "sources")
            node_targets[node] = self.find_total(node, "targets")

        max_path_long = 0
        node_source = 0
```

```
        node_target = 0
        for x in self.nodes:
            for y in self.nodes:
                if x == y:
                    continue

                if self.get_weight(x, y) == np.inf:
                    continue

                total_path_long = node_sources[x] + node_targets[y]
                if total_path_long > max_path_long:
                    max_path_long = total_path_long
                    node_source = x
                    node_target = y

        return {
            "source": node_source,
            "target": node_target
        }

    def find_total(self, node, to_find="sources"):
        sources = 0
        vis = [False for i in self.nodes]
        Q = deque([node])
        vis[node] = True

        while len(Q) > 0:
            x = Q.popleft()

            list = self.source[self.target == x] if to_find == 'sources' else
                self.target[self.source == x]

            for z in list:
                if not vis[z]:
                    vis[z] = True
                    Q.append(z)
                    sources += 1

        return sources
```

Código 7.8 Función complementaria para grafos 2.

```
#Estos codigos han sido extraidos de la pagina web: http://micaminomaster.com.
    co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/

import numpy as np
import math

from graph.DynamicGraph import DynamicGraph


class DynamicIncrementalGraph(DynamicGraph):

    def __init__(
        self,
        source=[],
```

```
        target=[],
        weight=[],
        directed=True,
        set_nodes_with_num_nodes=0
    ):
        self.set_action_to_graph = True
        DynamicGraph.__init__(self, source, target, weight, directed, set_nodes
            _with_num_nodes)
        self.clean_vars()

    def action_incremental(self, type_incremental, set_action_to_graph=True):
        self.set_action_to_graph = set_action_to_graph

        if type_incremental == "decrease_worst_edge":
            d = self.decrease_worst_weight()
        elif type_incremental == "insert_worst_edge":
            d = self.insert_worst_edge()
        elif type_incremental == "decrease_edge":
            d = self.decrease_random_weight()
        else:
            # type_incremental == insert_edge
            d = self.insert_random_edge(weights=[1])

        return d

    def reverse_action_incremental(self):
        if self.last_edge_action == "ADD":
            n1 = self.last_edge_updated[0]
            n2 = self.last_edge_updated[1]

            idx = np.where(np.logical_and(self.source == n1, self.target == n2))
                [0][0]

            self.source = np.delete(self.source, [idx])
            self.target = np.delete(self.target, [idx])

        self.clean_vars()

    def insert_edge(self, source, target, weight=1):
        self.clean_vars()

        last_edge_updated = np.array([source, target, weight])
        last_edge_action = "ADD"

        if self.set_action_to_graph:
            self.source = np.append(self.source, source)
            self.target = np.append(self.target, target)

            if len(self.weight) != 0:
                self.weight = np.append(self.weight, weight)

            self.last_edge_updated = last_edge_updated
            self.last_edge_action = last_edge_action

            self.sort_sources()

        return {
```

```
            "last_edge_updated": last_edge_updated.tolist(),
            "last_edge_action": last_edge_action
        }

    def insert_worst_edge(self, weight=1):
        self.clean_vars()

        result_found = self.find_source_target_worst_scenary_incremental_edge()
        source = result_found['source']
        target = result_found['target']

        return self.insert_edge(source, target, weight)

    def insert_random_edge(self, weights=[1, 2, 3, 4, 5, 6, 7, 8, 9]):

        count_max = 100
        flag = 0
        while True:
            source = np.random.choice(self.nodes, 1)[0]
            index_for_target = np.invert(np.logical_or(np.in1d(self.nodes, self.
                target[source == self.source]), self.nodes == source))

            choisen = self.nodes[index_for_target]
            if choisen.size != 0:
                target = np.random.choice(choisen, 1)[0]
                break
            flag = flag + 1
            if flag >= count_max:
                return -2

        w = np.random.choice(weights)
        return self.insert_edge(source, target, w)

    def insert_node(self, node, sources, w_sources, targets, w_targets):
        self.clean_vars()
        if self.nodes[self.nodes == node].size > 0:
            return -2

        sources = np.array(sources)
        targets = np.array(targets)
        self.source = np.concatenate((self.source, sources, np.full(targets.
            size, node)))
        self.target = np.concatenate((self.target, np.full(sources.size, node),
             targets))
        self.weight = np.concatenate((self.weight, w_sources, w_targets))
        self.nodes = np.append(self.nodes, node)

        self.last_node_updated['node'] = node
        self.last_node_updated['source'] = sources
        self.last_node_updated['target'] = targets

        self.last_node_action = "ADD"

        self.sort_sources()

        return self.last_node_updated
```

```
    def insert_random_node(self, num_edges=1, weights=[1, 2, 3, 4, 5, 6, 7, 8,
        9]):
        self.clean_vars()
        node = np.amax(self.nodes) + 1

        number = math.ceil(num_edges/2)
        sources = np.random.choice(self.nodes, number, replace=False)
        w_sources = np.random.choice(weights, number)

        number = num_edges - number
        if number <= 0:
            targets = []
            w_targets = []
        else:
            targets = np.random.choice(self.nodes, number, replace=False)
            w_targets = np.random.choice(weights, number)

        return self.insert_node(node, sources, w_sources, targets, w_targets)

    def decrease_weight(self, source, target, weight=1):
        self.clean_vars()
        if weight > self.get_weight(source, target):
            return -2

        last_edge_updated = np.array([source, target, weight])
        last_edge_action = "DECREASE"

        if self.set_action_to_graph:
            if len(self.weight) != 0:
                self.weight[np.logical_and(self.source == source, self.target ==
                     target)] = weight
                if not self.directed:
                    self.weight[np.logical_and(self.source == target, self.target
                         == source)] = weight

            self.last_edge_updated = last_edge_updated
            self.last_edge_action = last_edge_action

            self.sort_sources()

        return {
            "last_edge_updated": last_edge_updated.tolist(),
            "last_edge_action": last_edge_action
        }

    def decrease_worst_weight(self):
        self.clean_vars()

        result_found = self.find_edge_worst_scenary_update_edge()
        source = result_found['source']
        target = result_found['target']

        weight_current = self.get_weight(source, target)
        weight_new = 0 if weight_current > 0 else -1

        return self.decrease_weight(source, target, weight=weight_new)
```

```
    def decrease_random_weight(self, weight=1):
        count_max = 100
        flag = 0
        while True:
            source = np.random.choice(self.nodes, 1)[0]
            index_for_target = np.logical_or(np.in1d(self.nodes, self.target[
                source == self.source]), self.nodes == source)
            choisen = self.nodes[index_for_target]

            if choisen.size != 0:
                target = np.random.choice(choisen, 1)[0]
                if source == target:
                    continue
                weight = 1 if len(self.weight) != 0 else 0
                break

            flag = flag + 1
            if flag >= count_max:
                return -2

        return self.decrease_weight(source, target, weight=weight)
```

Código 7.9 Función complementaria para grafos 3.

```
#Estos codigos han sido extraidos de la pagina web: http://micaminomaster.com.
    co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/
```

Código 7.10 Función complementaria para grafos 4.

```
#Estos codigos han sido extraidos de la pagina web: http://micaminomaster.com.
    co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np


class GraphPro:
    source = []
    target = []
    weight = []
    nodes = []

    undirected = 0

    def __init__(
        self,
        source=[],
        target=[],
        weight=[],
        directed=True,
        set_nodes_with_num_nodes=0
    ):
        self.directed = directed
        self.source = np.array(source)
        self.target = np.array(target)
```

```
        self.weight = np.array(weight)

        if self.directed is False:
            self.source = np.concatenate([self.source, self.target])
            self.target = np.concatenate([self.target, np.array(source)])
            self.weight = np.concatenate([self.weight, self.weight])

        self.sort_sources()

        if set_nodes_with_num_nodes == 0:
            self.set_nodes()
        else:
            self.set_nodes_with_num_nodes(set_nodes_with_num_nodes)

    def get_weight(self, n1, n2):
        edges_found = np.logical_and(self.source == n1, self.target == n2)
        if not max(edges_found):
            return np.inf

        if len(self.weight) == 0:
            return 1

        if n1 == n2:
            return 0
        w = self.weight[edges_found]

        return np.inf if w.size == 0 else w[0]

    def get_weight_idx(self, idx):
        if len(self.weight) == 0:
            return 1

        return self.weight[idx]

    def set_nodes(self):
        if len(self.source) == 0:
            return 0

        max_nodes = max(np.concatenate([np.unique(self.source), np.unique(self.
            target)]))
        self.nodes = np.arange(max_nodes+1)
        return self.nodes

    def set_nodes_with_num_nodes(self, num_nodes):
        self.nodes = np.arange(num_nodes)
        return self.nodes

    def sort_sources(self):
        idx = self.source.argsort()
        self.source = self.source[idx]
        self.target = self.target[idx]
        if len(self.weight) != 0:
            self.weight = self.weight[idx]

    def get_targets_from_source(self, source, return_weight=False):
        start = np.searchsorted(self.source, source, side='left')
        end = np.searchsorted(self.source, source, side='right')
```

```
        targets_result = self.target[start:end]
        if return_weight:
            if len(self.weight) == 0:
                return targets_result, [1] * len(targets_result)

            return targets_result, self.weight[start:end]

        return targets_result

    def draw(self, with_weight=True):
        f = plt.figure()
        gr = nx.DiGraph()
        gr.add_weighted_edges_from(self.export())

        list_edges = list(gr.edges())
        list_nodes = list(gr.nodes())
        last = ()
        last_nodes = []

        if self.last_edge_updated.size > 0:
            last = (int(self.last_edge_updated[0]), int(self.last_edge_updated
                [1]))
            if last in list_edges:
                list_edges.remove(last)
                if not self.directed:
                    list_edges.remove((self.last_edge_updated[1], self.last_edge_
                        updated[0]))

        if self.last_node_updated is not None:
            for source in self.last_node_updated['source']:
                edge = (int(source), int(self.last_node_updated['node']))
                if edge in list_edges:
                    list_edges.remove(edge)
                last_nodes.append(edge)
            for target in self.last_node_updated['target']:
                edge = (int(self.last_node_updated['node']), int(target))
                if edge in list_edges:
                    list_edges.remove(edge)
                last_nodes.append(edge)
            if self.last_node_updated['node'] in list_nodes:
                list_nodes.remove(self.last_node_updated['node'])

        pos = nx.spring_layout(gr)
        nx.draw_networkx_edges(gr, pos=pos, with_labels=True, edgelist=list_
            edges, node_size=600)
        nx.draw_networkx_nodes(gr, pos=pos, with_labels=True, nodelist=list_
            nodes, node_size=600)

        if len(last) > 0:
            color = ''
            if self.last_node_action == "ADD":
                color = 'b'
            elif self.last_node_action == "DELETE":
                color = 'r'
            elif self.last_node_action == "UPDATE":
                color = 'g'
```

```
            if color != '':
                nx.draw_networkx_edges(gr, pos=pos, edgelist=[last], width=2.0,
                    edge_color=color)

        if len(last_nodes) > 0:
            color = ''
            color_node = ''
            if self.last_node_action == "ADD":
                color = 'b'
                color_node = 'b'

            if color != '':
                nx.draw_networkx_nodes(gr, pos=pos, with_labels=True, nodelist=[
                    self.last_node_updated['node']],
                                       node_color=color_node,
                                       node_size=600)
                nx.draw_networkx_edges(gr, pos=pos, edgelist=last_nodes, width
                    =2.0, edge_color=color)
                list_nodes.append(self.last_node_updated['node'])

        if with_weight:
            edge_labels = dict([((u, v,), d['weight']) for u, v, d in gr.edges(
                data=True)])
            nx.draw_networkx_edge_labels(gr, pos=pos, edgelist=list_edges, edge_
                labels=edge_labels)

        labels = dict()
        for i in list_nodes:
            labels[i] = str(i)


        nx.draw_networkx_labels(gr, pos, labels)
        f.savefig("graph.png", dpi=150)

        plt.show()
```

Código 7.11 Programa que ejecuta los diferentes procesos y crea la memoria compartida.

```
from  multiprocessing import shared_memory
import multiprocessing as mp
import numpy as np
import time
import cv2
import cv2.aruco as aruco
import modulo_imag
import interfaz_p
import keyboard

def actualiza_valor(nombre,nombre1):
  cap            =  cv2.VideoCapture(0)
  chess_size     =  [9,6]
  img_format     =  'png'
  aruco_dict     =  aruco.Dictionary_get(aruco.DICT_4X4_50)
  arucoParameters = aruco.DetectorParameters_create()
  ret, mtx, dist =  modulo_imag.calibrar(chess_size, img_format)
  #memoria compartida Video
  shm_video_scr  =  shared_memory.SharedMemory(name=nombre)
  video_scr      =  np.ndarray((480,640 , 3), dtype=np.int8, buffer=shm_video_
      scr.buf)

  #memoria compartida INFO
  shm_info       =  shared_memory.SharedMemory(name=nombre1)
  info           =  np.ndarray((6,), dtype=np.float64, buffer=shm_info.buf)

  while True:
    video_scr[:,:,:],info[:] = modulo_imag.pos_imag(aruco_dict,arucoParameters,
        ret ,mtx ,dist,cap)

    if keyboard.is_pressed('q'):
      print('se presionó [q]arar!')
      break
  return

def encender_pantalla(nombre,nombre1):
  interfaz_p.main(nombre,nombre1)
  return



if __name__ == '__main__':
  dtype_video   =  np.uint8
  shape_video   =  ((480,640 , 3))
  size_video    =  480*640*3 #bytes de informacion
  #Memoria compartida
  shm_video     =  shared_memory.SharedMemory(name=None,create=True, size=size_
      video)
  nombre        =  shm_video.name
  video         =  np.ndarray(shape_video, dtype=dtype_video, buffer=shm_video.
      buf)

  #Memoria compartida
  shm_INFO      =  shared_memory.SharedMemory(name=None,create=True, size=48)
  nombre1       =  shm_INFO.name
```

```
  INFO          =  np.ndarray((6,), dtype=np.float, buffer=shm_INFO.buf)

  #Creacion de procesos he inicialización
  proceso_1     =  mp.Process(target=actualiza_valor, args=(nombre,nombre1,))
  proceso_2     =  mp.Process(target=encender_pantalla, args=(nombre,nombre1,))

  proceso_1.start()
  time.sleep(2)

  proceso_2.start()
  time.sleep(10)

  # cerramos el bloque de memoria
  if True:
    del video
    shm_video.close()
    shm_video.unlink()
    shm_INFO.close()
    shm_INFO.unlink()
```

Código 7.12 Programa que crea la interfaz.

```
import random, pygame, sys
from  pygame.locals import *
from  multiprocessing import shared_memory
import time
import cv2
import numpy as np
from  PIL import Image
# Declaración de constantes y variables
BLACK = (0 , 0 , 0)
WHITE = (255 , 255 , 255)
alpha = (201,200,5)

def rotar(img,angulo,img_center):
  imgr          = pygame.transform.rotate( img, angulo )
  imgr_rec      = imgr.get_rect()
  imgr_rec.center = img_center
  return imgr,imgr_rec

def f_IA(roll,pitch,sky,IA,screen):
  pxy_IA        = 970 , 20
  center_sky    = 465 , 349+pitch*10
  skyr, skyr_rec = rotar(sky,roll,center_sky)
  IA.blit(skyr, skyr_rec)
  screen.blit(IA,pxy_IA)
  return

def f_brujula(yaw,brujula,screen):
  pxy_brujula         = 805 , 744
  center              = 960 , 899
  brujular,brujular_rec = rotar(brujula,yaw,center)
  screen.blit(brujular,brujular_rec)
  return

def texto_pos(screen,x,y,z):
```

```
  letra30                     = pygame.font.SysFont("Arial", 30)
  imagenTextoPresent          = letra30.render('X = '+str(int(x))+', '+'Y = '+
      str(int(y))+', '+'Z = '+str(int(z)),True, (200,200,200), (0,0,0) )
  rectanguloTextoPresent      = imagenTextoPresent.get_rect()
  rectanguloTextoPresent.centerx = 200
  rectanguloTextoPresent.centery = 900
  screen.blit(imagenTextoPresent,rectanguloTextoPresent)

def main(nombre,nombre1):
  #Se inicializa la camara (en caso de ya estarla que es el caso se intentara
      pasar cap a main)

  yaw   = 127
  pitch = 10
  roll  = 20
  #Memoria compartida
  shm_video_scr = shared_memory.SharedMemory(name=nombre)
  video_scr     = np.ndarray((480,640 , 3), dtype=np.int8, buffer=shm_video_scr.
      buf)

  #Memoria compartida angulos
  shm_info = shared_memory.SharedMemory(name=nombre1)
  info     = np.ndarray((6,), dtype=np.float64, buffer=shm_info.buf)
  frame    = np.zeros(shape=(480,640,3),dtype=np.uint8)

  # Se inicializa el juego
  pygame.init()
  pygame.display.set_caption("Datos")
  screen_size=1920,1080
  screen = pygame.display.set_mode(screen_size,pygame.FULLSCREEN)

  #se cargan las imagenes
  #inicio
  inicio = pygame.image.load('iniciacion2.png').convert()
  screen.blit(inicio,(-1,0))
  pygame.display.flip()
  time.sleep(10)
  #Fondo
  fondo = pygame.image.load('fondo.png').convert()
  fondo.set_colorkey(BLACK)
  #Horizonte artificial
  sky   = pygame.image.load('sky2.png').convert()

  #Creamos una superficie donde subir sky con dimensiones correctas
  IA_size = (930,698)
  IA      = pygame.Surface(IA_size)
  IA.fill(WHITE)

  #Brujula
  brujula = pygame.image.load('brujula.png').convert()
  brujula.set_colorkey(BLACK)



#   centro_dial = 250,250
#   centro_sky = 250,250+pitch*6
  # Bucle principal
```

```
  currTime = time.time()


  while True:
    [yaw,pitch,roll] = info[0:3]
    roll           = roll-90
    [x,y,z]        = info[3:6]

    ,,,
    newTime=time.time()
    if (newTime-currTime)> 1 :
      yaw=random.randint(0, 360)
      pitch=random.randint(-30, 30)
      roll=random.randint(-30,30)
      currTime=newTime
    ,,,
    #leemos y acondicionamos frames de la camara
    frame[:,:,:]     =  video_scr[:,:,:]
    frame2           = cv2.resize(frame, (930, 697))
    frame2           = cv2.cvtColor(frame2,cv2.COLOR_BGR2RGB)
    video            = pygame.pixelcopy.make_surface(frame2)
    videor, videor_rec = rotar(video,-90,(485 , 368.5))

    # 1.- Se dibuja la pantalla
    #se da color negro de base y sirve para borrar lo q hay en pantalla
    screen.fill(BLACK)

    #se imprime el horizonte artificial
    f_IA(roll,pitch,sky,IA,screen)

    #se imprime la camara y el fondo
    screen.blit(videor,videor_rec)
    screen.blit(fondo,(0 , 0))

    #y por ultimo se imprime la brujula
    f_brujula(yaw,brujula,screen)

    #Imprimimos datos de pos
    texto_pos(screen,x,y,z)
    #actaulizamos ventana
    pygame.display.flip()
    # 2.- Se comprueban los eventos
    for event in pygame.event.get():
      if event.type == QUIT:
        exit()
      if event.type == KEYDOWN :
        if event.key == K_q:
            # cerramos el bloque de memoria
          if False:
            del video_scr # Unnecessary; merely emphasizing the array is no
                longer used
            shm_video_scr.close()
            shm_video_scr.unlink()
          exit()
# Este fichero es el que ejecuta el juego principal
```

Código 7.13 Programa que suministra la información de vídeo y detección de marcadores a la interfaz.

```
import random, pygame, sys
from  pygame.locals import *
from  multiprocessing import shared_memory
import time
import cv2
import numpy as np
from  PIL import Image
# Declaración de constantes y variables
BLACK = (0 , 0 , 0)
WHITE = (255 , 255 , 255)
alpha = (201,200,5)

def rotar(img,angulo,img_center):
  imgr          = pygame.transform.rotate( img, angulo )
  imgr_rec      = imgr.get_rect()
  imgr_rec.center = img_center
  return imgr,imgr_rec

def f_IA(roll,pitch,sky,IA,screen):
  pxy_IA        = 970 , 20
  center_sky    = 465 , 349+pitch*10
  skyr, skyr_rec = rotar(sky,roll,center_sky)
  IA.blit(skyr, skyr_rec)
  screen.blit(IA,pxy_IA)
  return

def f_brujula(yaw,brujula,screen):
  pxy_brujula         = 805 , 744
  center              = 960 , 899
  brujular,brujular_rec = rotar(brujula,yaw,center)
  screen.blit(brujular,brujular_rec)
  return

def texto_pos(screen,x,y,z):
  letra30                     = pygame.font.SysFont("Arial", 30)
  imagenTextoPresent          = letra30.render('X = '+str(int(x))+', '+'Y = '+
      str(int(y))+', '+'Z = '+str(int(z)),True, (200,200,200), (0,0,0) )
  rectanguloTextoPresent      = imagenTextoPresent.get_rect()
  rectanguloTextoPresent.centerx = 200
  rectanguloTextoPresent.centery = 900
  screen.blit(imagenTextoPresent,rectanguloTextoPresent)

def main(nombre,nombre1):
  #Se inicializa la camara (en caso de ya estarla que es el caso se intentara
      pasar cap a main)

  yaw   = 127
  pitch = 10
  roll  = 20
  #Memoria compartida
  shm_video_scr = shared_memory.SharedMemory(name=nombre)
  video_scr     = np.ndarray((480,640 , 3), dtype=np.int8, buffer=shm_video_scr.
      buf)

  #Memoria compartida angulos
```

```
  shm_info = shared_memory.SharedMemory(name=nombre1)
  info    = np.ndarray((6,), dtype=np.float64, buffer=shm_info.buf)
  frame   = np.zeros(shape=(480,640,3),dtype=np.uint8)

  # Se inicializa el juego
  pygame.init()
  pygame.display.set_caption("Datos")
  screen_size=1920,1080
  screen = pygame.display.set_mode(screen_size,pygame.FULLSCREEN)

  #se cargan las imagenes
  #inicio
  inicio = pygame.image.load('iniciacion2.png').convert()
  screen.blit(inicio,(-1,0))
  pygame.display.flip()
  time.sleep(10)
  #Fondo
  fondo = pygame.image.load('fondo.png').convert()
  fondo.set_colorkey(BLACK)
  #Horizonte artificial
  sky   = pygame.image.load('sky2.png').convert()

  #Creamos una superficie donde subir sky con dimensiones correctas
  IA_size = (930,698)
  IA     = pygame.Surface(IA_size)
  IA.fill(WHITE)

  #Brujula
  brujula = pygame.image.load('brujula.png').convert()
  brujula.set_colorkey(BLACK)



#   centro_dial = 250,250
#   centro_sky = 250,250+pitch*6
  # Bucle principal
  currTime = time.time()


  while True:
    [yaw,pitch,roll] = info[0:3]
    roll           = roll-90
    [x,y,z]        = info[3:6]

    '''
    newTime=time.time()
    if (newTime-currTime)> 1 :
      yaw=random.randint(0, 360)
      pitch=random.randint(-30, 30)
      roll=random.randint(-30,30)
      currTime=newTime
    '''
    #leemos y acondicionamos frames de la camara
    frame[:,:,:]     =  video_scr[:,:,:]
    frame2           = cv2.resize(frame, (930, 697))
    frame2           = cv2.cvtColor(frame2,cv2.COLOR_BGR2RGB)
    video            = pygame.pixelcopy.make_surface(frame2)
```

```
    videor, videor_rec = rotar(video,-90,(485 , 368.5))

    # 1.- Se dibuja la pantalla
    #se da color negro de base y sirve para borrar lo q hay en pantalla
    screen.fill(BLACK)

    #se imprime el horizonte artificial
    f_IA(roll,pitch,sky,IA,screen)

    #se imprime la camara y el fondo
    screen.blit(videor,videor_rec)
    screen.blit(fondo,(0 , 0))

    #y por ultimo se imprime la brujula
    f_brujula(yaw,brujula,screen)

    #Imprimimos datos de pos
    texto_pos(screen,x,y,z)
    #actaulizamos ventana
    pygame.display.flip()
    # 2.- Se comprueban los eventos
    for event in pygame.event.get():
      if event.type == QUIT:
        exit()
      if event.type == KEYDOWN :
        if event.key == K_q:
            # cerramos el bloque de memoria
          if False:
            del video_scr # Unnecessary; merely emphasizing the array is no
                longer used
            shm_video_scr.close()
            shm_video_scr.unlink()
          exit()
# Este fichero es el que ejecuta el juego principal
```

Índice de Figuras

Índice de Códigos

Bibliografía

[1] Proyect AiRT, *https://airt.webs.upv.es/*.

[2] España Deloitte, *Forces of change: Industry 4.0*.

[3] AiRT Proyect, *https://www.prosegur.com/media/articulo/prensa/licencia-aesa-para-volar-drones*.

https://www2.deloitte.com/es/es/pages/manufacturing/articles/que-es-la-industria-4.0.html

https://airt.webs.upv.es/

https://www.prosegur.com/media/articulo/prensa/licencia-aesa-para-volar-drones

https://ieeexplore.ieee.org/abstract/document/7060497

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.9094&rep=rep1&type=pdf

https://www.researchgate.net/profile/Samer-Saab-3/publication/224153238_A_Standalone_RFID_Indoor_Positioning_System_Using_Passive_Tags/links/0c960536bae690da36000000/A-Standalone-RFID-Indoor-Positioning-System-Using-Passive-Tags.pdf

https://ipsjcva.springeropen.com/articles/10.1186/s41074-017-0027-2

https://www.automate.org/blogs/what-is-visual-slam-technology-and-what-is-it-used-for

https://es.wikipedia.org/wiki/Sensor_CMOS

https://es.wikipedia.org/wiki/Dispositivo_de_carga_acoplada

https://www.unmannedsystemstechnology.com/category/supplier-directory/cameras-imaging-systems/cameras/

https://z3technology.com/

https://www.raptorphotonics.com/products/cmos-oem-solutions/

https://www.hamamatsu.com/eu/en/product/cameras/cmos-cameras/index.html

https://thinklucid.com/downloads-hub/

https://www.sony-semicon.co.jp/e/products/IS/camera/product.html

https://unipython.com/calibracion-la-camara-opencv/

https://www.iberdrola.com/innovation/what-is-augmented-reality

https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html

https://www.forbes.com/sites/bernardmarr/2018/07/30/9-powerful-real-world-applications-of-augmented-reality-ar-today/?sh=14af36382fe9

https://www.uco.es/investiga/grupos/ava/node/26

https://www.uco.es/investiga/grupos/ava/node/68

https://chev.me/arucogen/

https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit

https://docs.opencv.org/4.5.0/d5/dae/tutorial_aruco_detection.html

https://docs.opencv.org/4.5.0/d9/d6a/group__aruco.html

http://micaminomaster.com.co/grafo-algoritmo/algoritmos-dijkstra-floyd-warshall-python-3-5/

https://aprendeconalf.es/docencia/python/manual/pandas

https://docs.python.org/3/contents.html

https://www.pygame.org/docs/

https://www.skybrary.aero/index.php/Flight_Instrument_Presentation_of_Aircraft_Attitude

https://docs.python.org/3/library/multiprocessing.html

https://docs.python.org/3/library/multiprocessing.shared_memory.html